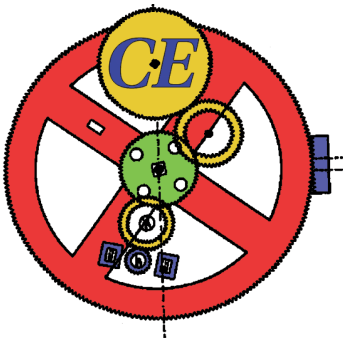

MSc Thesis

FPGA accelerated Facial Recognition

Nikolaos Stekas

ES-MS-4415833

Abstract

The ability to recognize faces is highly important in many areas of development. Though the years, the evolving technologies, enabled this process to be adapted in modern computer systems. These systems can be found in a wide variety of areas that yield significant impact. Therefore, there is an increasing demand for fast and accurate systems, able to perform facial recognition.
In this thesis, a face recognition implementation on a FPGA-based System on Chip (SoC), is presented. This implementation utilizes Local Binary Patterns Histograms to extract features from test face images and Manhattan Distance to retrieve the correct match from the systems face database. The SoC utilized is a Zynq-7030. The feature extraction and the distance computations, between the database, are implemented on the FPGA. The ARM processor of the SoC is responsible for receiving the input stream and presenting the output result, using the acquired distances. Real-time, high accuracy face recognition, with an execution time of 2.4 ms and accuracy of 78%, is achieved through this implementation.

TU Delft
Delft University of Technology

Faculty of Electrical Engineering, Mathematics and Computer Science

FPGA accelerated Facial Recognition

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

EMBEDDED SYSTEMS

by

Nikolaos Stekas
born in Volos, Greece

Computer Engineering
Department of Electrical Engineering
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

FPGA accelerated Facial Recognition

by Nikolaos Stekas

Abstract

The ability to recognize faces is highly important in many areas of development. Though the years, the evolving technologies, enabled this process to be adapted in modern computer systems. These systems can be found in a wide variety of areas that yield significant impact. Therefore, there is an increasing demand for fast and accurate systems, able to perform facial recognition.

In this thesis, a face recognition implementation on a FPGA-based System on Chip (SoC), is presented. This implementation utilizes Local Binary Patterns Histograms to extract features from test face images and Manhattan Distance to retrieve the correct match from the systems face database. The SoC utilized is a Zynq-7030. The feature extraction and the distance computations, between the database, are implemented on the FPGA. The ARM processor of the SoC is responsible for receiving the input stream and presenting the output result, using the acquired distances. Real-time, high accuracy face recognition, with an execution time of 2.4 ms and accuracy of 78%, is achieved through this implementation.

Laboratory : Computer Engineering
Codenumber : ES-MS-4415833

Committee Members :

Advisor:	dr.ir. Stephan Wong, CE, TU Delft
Chairman:	dr.ir. Stephan Wong, CE, TU Delft
Member:	dr.ir. Arjan van Genderen, CE, TU Delft
Member:	dr.ir. Rene van Leuken, CAS, TU Delft
Member:	Dirk van den Heuvel, Topic

"It is the mark of an educated mind to be able to entertain a thought without accepting it."

- Aristotle

Contents

List of Figures

List of Tables

List of Acronyms

ASIC Application Specific Integrated Circuit

AXI Advanced eXtensible Interface

BRAM Block Random Access Memory

DMA Direct Memory Access

FF Flip-Flop

FPGA Field Programmable Gate Array

GPU Graphics Processing Unit

IP Intellectual Property

LBPH Local Binary Patterns Histograms

LUT Look Up Table

PL Programmable Logic

SoC System on Chip

TDK Topic Development Kit

UFI Unconstrained Facial Images

Acknowledgements

Foremost, I would like to express my sincere gratitude to my parents, Dimitrios and Garyfallia, and the rest of my family for their support, during all the years of my studies. I would also like to thank my friends, that encouraged me throughout the whole thesis and assisted me, whenever needed, on different tasks.

I also owe thanks to my supervisor dr. ir. Stephan Wong, for his valuable help and guidance, during my thesis. In addition, I would like to show my appreciation to my other professors in TU Delft, for the knowledge and experience I acquired during my studies.

Last but not least, I would like to thank wholeheartedly Topic Embedded Systems and all the people there, for providing me with all the necessities, in order to complete my work successfully. I should, moreover, offer thanks to Dirk van den Heuvel, who was my supervisor on behalf of Topic and helped me achieve my goals with his ideas and experience.

Nikolaos Stekas
Delft, The Netherlands
May 30, 2016

1 Introduction

Facial Recognition is a subject that researchers have focused on for years. In the current Master of Science thesis, a real-time facial recognition system will be implemented, exploiting FPGA technology. This project was carried out in collaboration with TOPIC Embedded Systems, that set the specifications of the project and provided all resources needed to proceed with the implementation. In this chapter, in Section 1.1 the motivation around this project is stated, in Section 1.2 a description about the project and its specifications are given, in Section 1.3 the goals of this project are stated and finally in Section 1.4 the outline of the Thesis is described.

1.1 Motivation of Work

Humans mainly use faces to recognize individuals. In the past decades, the advances in technology enabled this recognition to take place, similarly, in computing systems. The first implementations [1], [2] could be characterized as semi-automated. The required features for recognition, were located and marked manually and then compared by the computer system. In the future years, completely automated simple geometrical models, based on linear algebra models, were utilized to perform recognition. Such models are described in Chapter 3 ([3], [4]). Nowadays, sophisticated models are utilized for the same purpose. Those models utilize geometrical features of the face in combination with advanced mathematical models ([5], [6], [7]), increasing this way the accuracy of the process significantly.

Aside from the accuracy, execution time of facial recognition processes is also of high importance. The newly introduced, complex, models demand intensive computations that consume a notable amount of time. New technologies in the field of processors made it possible to accelerate the facial recognition process, achieving, presently, real-time demands ([8], [9], [10]).

This evolution in the field of facial recognition, enabled solutions in a number of different sections varying from medical assistance to security systems. The introduction of face recognition in these areas of interest, combined to the wide spreading into simple industry solutions, highlighted the importance of the subject and its mandatory evolution. Thus, the bar for face recognition technologies is raised constantly, with a need on implementing systems that are faster and of higher accuracy.

The current project aims to cope with those demanding needs and derive effective solutions. Taking advantage of contemporary technology, FPGAs in specific, will be the main aid towards this goal.

1.2 Project Definition

The present project is aiming on designing a facial recognition system, in an embedded platform utilizing FPGA technology. The basic idea, is to utilize a camera and recognize the face that is being recorded. A database that has been stored in the platform will be used for this purpose.

In more detail, a camera recording at a rate of 60 fps will be used to import images. Those images will first go through a face detection process, that has been already implemented separately, and gives one or multiple faces as an output, in a specific image size, 200x200 pixels.

After this process is complete, the current project part takes over. The face that has been detected, must be recognized. An algorithm is first used to extract features out of the recorded face. Then, the matching process can take place. A database of faces is required for this. We should mention here that the creation of the database has been done beforehand and the computation intensity to create it is not of our concern and is not needed to be taken in consideration. The total number of faces in the database will be 100, and a total of 20 different people(5 images per person), as specified for the project. Also, as specified for this project, we assume that the face detected is of someone included in the database and not an unknown individual. This way, the best match is recognized as the recorded person. The last two specifications were set by Topic and are derived by the final product needs. The overview of the project can be viewed in Figure 1.1

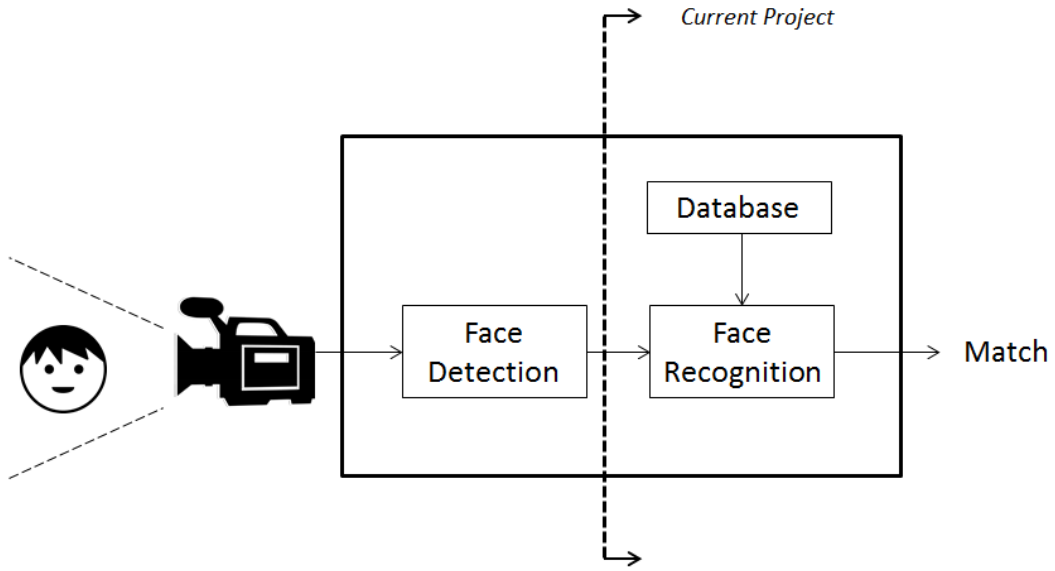

Figure 1.1: *General overview of the project.*

The embedded platform utilized includes a heterogeneous processor (Zynq-7030) consisting of both a CPU and a FPGA. The implementation will exploit both features, but

will focus on the FPGA processor in order to carry out the compute-intensive parts, achieving this way a real-time system responding to the frame rate of the camera (60fps).

1.3 Project Goals

The main goal of this project is to implement a face recognition system utilizing FPGA technology. Accuracy and execution time are the main aspects, that focus is needed on, in order to achieve real-time, high accuracy face recognition. Real-time performance is met, if the recognition is performed in less than 16 ms. This time is derived by the frame rate of the camera. As 60 frames must be processed at 1 second, each frame must be processed in less than 16 ms ($1/60 = 0.016$ seconds) . In addition, high-accuracy is translated to a first rank accuracy percentage of higher than 75%, matching the accuracy achieved by the state of the art models, that will be described in Chapter 3.

To achieve this, research regarding facial recognition algorithms and implementation needs to be carried out. Through this research, an effective algorithm, that can provide high accuracy in a manageable time, needs to be chosen to apply it on our system and also identify how other completed implementations perform. Research is also needed concerning the implementation process, this way the proper path for reaching our goal effectively can be approximated and followed during our implementation design. The goals of this project can be summarized to the following:

- Research on facial recognition algorithms and conclude on the most suitable for our system.
- Research on other implementations of the algorithm and investigate on their performance and efficiency.
- Analyze the algorithm and select the proper parameters for better efficiency.
- Examine the role of the FPGA, to efficiently accelerate the chosen model.
- Implement the design for the system specified.
- Achieve real-time face recognition, with an execution time below 16 ms.
- Obtain high-accuracy, which is equal or higher than 75%, analogous to the accuracy of the state of the art recognition models.

1.4 Thesis Outline

The thesis is organized in the following way. Chapter 2, includes background information regarding this project, helping with its comprehension. In Chapter 3, facial recognition algorithms are presented and a comparison between them takes place in order to determine the most suitable one for our implementation. Next, other implementations of the chosen algorithm on different platforms are presented and their performance is being examined. In Chapter 4, an analysis of the algorithm development takes place. This analysis regards the selection of the most appropriate models and parameters for the

algorithm in order to achieve the best possible result. In Chapter 5, the implementation, itself, is being described thoroughly, from the implementation approach to the last design details . Next, in Chapter 6, experimental results of the implemented design are examined and compared to similar implementations. Finally, in Chapter 7, a summary of our work is given and future work is proposed.

2 Background

Background information relevant to this work will be described in this chapter. The main fields combined in this work are facial recognition and reconfigurable computing. In Section 2.1 an introduction on the first field will be presented. Next, in Section 2.2 some basic concepts of reconfigurable computing will be described. In Section 2.3, the platform utilized in this work, combining the two fields, will be introduced. In the end, in Section 2.4, a conclusion concerning the background information will be presented.

2.1 Facial Recognition Systems

A facial recognition system is a computer application, that its purpose is identifying an input face successfully. In Section 1.1, the evolution on the field has been presented, leading to today's complex, but efficient models. Those models, alongside with present accelerated face recognition implementations, will be described, in more detail, in Chapter 3.

Despite the differences (based-model, input form, etc.) every facial recognition system is based on the same outline as follows: Import an input face, extract a set of facial features, match those features to the system's facial features database and finally recognize the face according to the best matching. This procedure can be observed in Figure 2.1.

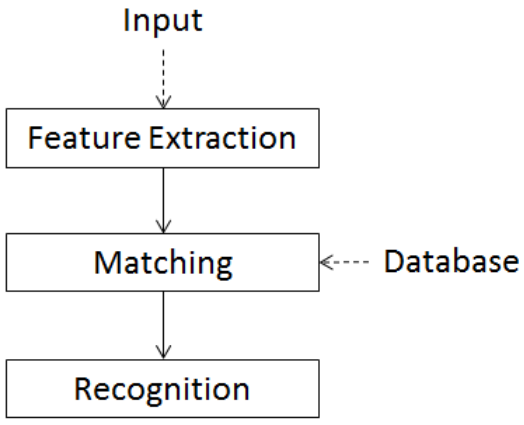

Figure 2.1: *The procedure of face recognition.*

In the first part of the system, the facial features, that are going to be extracted, and the process of this extraction, need to be decided. Those features should lead to accurate facial recognition, by being as unique as possible for every different face.

For example, the size of the face is not a proper feature, as many faces may have the same size. Moreover, these features should be tolerant to different conditions such as environmental conditions, e.g. lighting, expressions, etc. The process for doing that should be also designed carefully, in order to decrease complexity and maintain a high performance.

Once the facial features, the system utilizes, are decided, a database is constructed. This database consists of the facial features of a number of different persons. Those persons are the ones intended to be identified through the system. The system's memory should be taken in account, so that the size of the database is within its limits.

Next, the matching phase should be designed. The facial features acquired by the input face, need to be compared to all the facial features of the database. As those feature are represented by values in computer applications, distance measures are utilized for this purpose. At the end of this process, a distance is calculated between the test face and each face of the database.The recognition is then finalized by returning the face identification of the minimum distance, also called best match.

Concerning the input and the output of a facial recognition system, they are completely depended on the purpose of the system utilization. The input may vary from continuous video frames, to face images imported from a large database. Same for the output, where a face picture of the matching person or just an identity word (e.g. person's name) might be returned.

In conclusion, the same process is the basis behind every facial recognition systems. Every step of this process differs between different application and should be examined carefully depending on the systems specifications.

2.2 Reconfigurable Computing

The subject of reconfigurable computing will be examined in this section, as it is the basis for the current system design.

In traditional computing, two ways are met for computing algorithms [11]: ASIC (Application Specific Integrated Circuit) and software-programmed processors.

ASICs are used to perform the operations in hardware, that has been designed specifically for the current algorithm. Because of this specific design, the execution time for an algorithm reduces significantly, leading to faster implementations. Therefore, ASICs are proven to be really efficient. The drawback of this determinate design, is that no alteration is feasible after fabrication, making ASICs inflexible.

Processors on the other hand, are characterized by flexibility. The change of instructions leads to a different system functionality, on the same hardware. The cost of this flexibility, though, is much slower execution. The processor must first read each instruction from the memory, translate it, and only then proceed with its execution. This process is resulting in a high execution overhead for each operation.

A need for systems that combine the flexibility of processors and the high performance of ASICs is obvious. Reconfigurable computing is intended to merge those beneficial aspects together. Reconfigurable devices can be configured specifically to execute complex combinational functions, like ASICs, and in addition, this configuration may be altered later on time, providing flexibility, like processors.

The most common reconfigurable devices are the Field-Programmable Gate Arrays (FPGAs). Every FPGA consists of a finite number of predefined resources connected with programmable interconnects to implement a reconfigurable digital circuit and I/O blocks to allow the circuit to access the outside world [12], as seen in Figure 2.2.

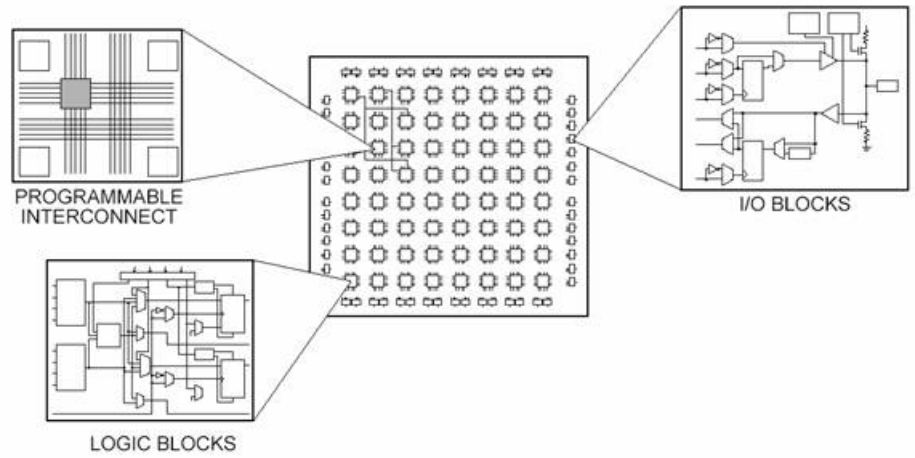

Figure 2.2: *FPGA overview. [12]*

Logic blocks, also referred as configurable logic blocks (CLBs), are the basic logic unit of the FPGAs. Those blocks are made up of two primary components: Flip-Flops (FFs) and Look-Up Tables (LUTs). LUTs are a set of gates that are combined together to create an entity of K inputs. This entity is able to perform any K-input function effectively [13], and is therefore the prime unit on FPGA implementations.

FPGAs have evolved radically in the past years. More resources available for configuration are presently included in the FPGAs. Moreover, partial reconfiguration (reconfiguring only a part of the FPGA while the rest is operating) is nowadays feasible. Aside from that, FPGAs have been integrated into platforms and Systems on Chips (SoCs), to exploit their performance, as the benefits derived from their utilization are more and more needed in modern applications.

2.3 Topic Development Kit

The integration of FPGAs in complete platforms is common nowadays, as mentioned in Section 2.2. A platform of this type will be utilized for the purposes of this project. This platform is the Topic Development Kit [14] developed by Topic Embedded Systems.

The main component of the platform is the Miami System-on-Module (SoM) assembled on the Florida carrier board, both developed by Topic Embedded Systems. The specifications of the Miami SoM are the primary concern of the current project. The Florida carrier aims on enabling the platform for multiple kinds of development, including many communication interfaces. This is useful for creating practical interaction extensions, but should not concern as in the present.

The Miami SoM includes a Z-7030 SoC from the Zynq®-7000 series All Programmable SoCs [15]. Regarding other main features, a DDR3 memory of 512MB is

available and 140 pins for interface connections such as UART, JTAG, etc. In more detail, the Z-7030 includes an ARM Cortex-A9 with two cores at 800Mhz, and an FPGA, Kintex®-7. All specifications of the the Z-7030 are given in detail in Figure 2.3, for a better overview of the platform. The available resources will be taken more carefully in consideration, during the implementation stages, to achieve the best result. However, in general the resources, as presented, indicate a powerful system that can help towards the project goal of real-time face recognition. Most of those

	Device Name	Z-7030
	Part Number	XC7Z030
Processing System	Processor Core	Dual ARM® Cortex™-A9 MPCore™ with CoreSight™
	Processor Extensions	NEON™ & Single / Double Precision Floating Point for each processor
	Maximum Frequency	866MHz Up to 1GHz(1)
	L1 Cache	32KB Instruction, 32KB Data per processor
	L2 Cache	512KB
	On-Chip Memory	256KB
	External Memory Support(2)	DDR3, DDR3L, DDR2, LPDDR2
	External Static Memory Support(2)	2x Quad-SPI, NAND, NOR
	DMA Channels	8 (4 dedicated to Programmable Logic)
	Peripherals	2x UART, 2x CAN 2.0B, 2x I2C, 2x SPI, 4x 32b GPIO
	Peripherals w/ built-in DMA(2)	2x USB 2.0 (OTG), 2x Tri-mode Gigabit Ethernet, 2x SD/SDIO
	Security(3)	RSA Authentication of First Stage Boot Loader, AES and SHA 256b Decryption and Authentication for Secure Boot
	Processing System to Programmable Logic Interface Ports (Primary Interfaces & Interrupts Only)	2x AXI 32b Master, 2x AXI 32b Slave 4x AXI 64b/32b Memory AXI 64b ACP 16 Interrupts
Programmable Logic	7 Series Programmable Logic Equivalent	Kintex®-7 FPGA
	Logic Cells (Approximate ASIC Gates(4))	125K (~1.9M)
	Look-Up Tables (LUTs)	78,600
	Flip-Flops	157,200
	Total Block RAM (# 36Kb Blocks)	9.3Mb (265)
	Programmable DSP Slices (18x25 MACCs)	400
	Peak DSP Performance (Symmetric FIR)	593 GMACs
	PCI Express® (Root Complex or Endpoint)	Gen2 x4
	Analog Mixed Signal (AMS) / XADC(2)	2x 12 bit, MSPS ADCs with up to 17 Differential Inputs
	Security(3)	AES and SHA 256b Decryption and Authentication for Secure Programmable Logic Configuration

Figure 2.3: *Z-7030 specifications. [15]*

2.4 Conclusion

In the current chapter, a brief introduction on the background information, needed to comprehend our work, was presented. The basic outline of a facial recognition system was described in Section 2.1. Next, the field of reconfigurable computing was presented in 2.2 and in the end, in Section 2.3 our development kit is presented. The information shared, although it is on an introductory level, is helpful for comprehending the work carried out later.

3 Literature Review

In the current chapter the background of facial recognition models and implementations will be examined. Firstly, in Section 3.1, the most dominant algorithms for achieving recognition, the basis of facial recognition systems, are presented and compared. This way the most suitable algorithm will be chosen for our implementation. Second, in Section 3.2, implementations of the selected algorithm in different platforms will be shown and their performance will be discussed. Finally, in Section 3.3, conclusion for the current chapter will be derived.

3.1 Facial Recognition Algorithms

The groundwork for any face recognition system is the algorithm that is responsible for recognizing a face. As seen in Chapter 2.1, the process of recognizing a face is first extract features from a probe face and then, use this features to match this face to a face in the existing database. Therefore, we can distinguish two parts: features extraction and matching. All the recognition algorithms focus on the first part, as the matching can be done efficiently with presently existing distance measures for any of the features. Different kind of features have been proposed for efficient face recognition, from simple geometrical features, to high contrast values existence.

After a research around the subject a number of different algorithms that can be utilized were found. The most dominant ones should be examined and compared in order to conclude to the algorithm that our system is going to utilize. The algorithms that were found to be the most promising and are going to be examined are: Eigenfaces, Fisherfaces, Elastic Bunch Graph Matching (EBGM), Local Binary Patters Histograms (LBPH), Scale Invariant Feature Transform (SIFT).

3.1.1 Eigenfaces

The eigenfaces method for recognizing faces was introduced in 1991 [3] and it is based on principal component analysis (PCA) [16], where a set of possibly correlated values is converted to set of linearly uncorrelated values. This way the amount of values needed is decreased significantly. This method enabled for the first time reliable, automated, real-time face recognition systems. The basic idea of the method is to utilize the eigenvectors (eigenfaces) of faces, as the linearly uncorrelated values, to distinguish them.

More specifically, a number, K, of gray-scaled images of size MxN that will form the database are needed. Each of these images will be represented as a vector, I_K, of size $1x(MxN)$. Having all those images the mean image, AI, is calculated

$$AI = \frac{\sum_{k=1}^{K} I_k}{K}$$

and also the deviance for each of the images.

$$ID_K = I_K - AI$$

Next, the all the vectors, ID_K, are merged into matrix A of size $Kx(MxN)$.Matrix C is then calculated

$$C = AA^T$$

and the eigenvectors and eigenvalues of the matrix are calculated. A number, i, of eigenvectors, corresponding to the highest eigenvalues, depending on the size of information needed is maintained. Those eigenvectors, E_i, correspond to faces called eigenfaces. Such eigenfaces can be viewed in Figure 3.1

Figure 3.1: *Different eigenfaces corresponding to varying eigenvalues [17]*

The eigenfaces, E_i, are then used to find a set of weights, $w_{ki}$for each image.

$$w_{Ki} = I_K * E_i$$

These weights represent the contribution of each eigenface in forming the original image. So in the end each image is described by a set of weights.

A probe image, I_P, is then needed to be identified. The weights, w_{KP} ,of this image are calculated the same way. Finally, a distance measure is used (e.g. Euclidean Distance) to find the distance between the probe image's weights and the weights of each of the database images. After the distances between all images have been calculated, the one with the smaller number leads to the best face match and identification is completed.

The research indicates that 40 eigenfaces are enough to succeed with recognition. This way the database has to consist of 40 eigenfaces and 40 weights for each of the images that we want to include in. Also we can observe that for every probe image the computations are simple to lead to recognition, with the matrix multiplications being the most intensive part. Eigenfaces ,though, is not so effective under different light conditions and expressions. Generally, it performs well under controlled conditions which are difficult to get in everyday life environments.

3.1.2 Fisherfaces

Fisherfaces [4] is another popular face recognition algorithm. It was developed in 1997 and it is based on Fisher's Linear Discriminant [18], a method used to find a linear combination of features that characterizes classes of objects. Many similarities can be reported between Eigenfaces and Fisherfaces as both are based on linearly projecting.

More in detail, once again, a number, K, of gray-scaled images of size MxN, represented as vectors, I_K, that will form the database are used. In addition we categorize those images into a number of classes, c. This categorization is usually done by including the images of the same person in the same class. With all those images the average image, AI, is calculated

$$AI = \frac{\sum_{k=1}^{K} I_k}{K}$$

Also the average image, AI_c for each class is calculated

$$AI_c = \frac{\sum_{c=1}^{size} I_c}{K}$$

where $size$ is the size of the class and I_c are the images included in the class. Each class' mean image deviation from the mean image,CD_c, can now be calculated.

$$CD_c = AI_c - AI$$

The deviation of the images inside a class from the class' mean image, ID_c , is also calculated.

$$ID_c = I_c - AI$$

Two merged matrices are then created, A which includes CD_c for all classes and B which includes ID_c for all classes. Using A and B the relative eigenvectors, E and eigenvalues, l, are calculated.

$$(AA^T)E = l(BB^T)E$$

From this point on we proceed as in eigenfaces. A number, i, of eigenvectors depending on the eigenvalues is maintained. Those eigenvectors correspond to images called fisherfaces, Figure 3.2 shows such fisherface. The weights for each image are once more calculated.

$$w_{Ki} = I_K * E_i$$

Whenever a probe image is inserted the same recognition procedure is followed as in eigenfaces for calculating its weights and the image with the closest weight values is marked as the best match,completing the identification process.

A problem that might be encountered while using fisherfaces, is when the classes are more than the images. To override this issue, when fisherfaces is used, the sample vectors are projected onto the PCA space used in eigenfaces and the fisherfaces are then computed in this PCA.

A system utilizing fisherfaces will need the same size database as in eigenfaces, 40 fisherfaces and 40 weights per image in the database. Moreover, the recognition process calculations are the exact same as in eigenfaces. Creating the database is the only part

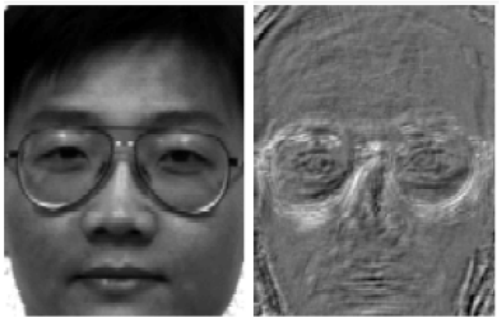

Figure 3.2: *Fisherface derived from a face image in the Yale Database [4]*

that requires more calculations, but as stated in Section 1.2, it should not concern the current project.

Regarding, the accuracy of the algorithm, we can observe that fisherfaces enables classification, which can improve accuracy noticeably. A database that includes a person's images under different lightning or expression conditions can make the algorithm insensitive to those changes. More images of a person under different conditions lead to better results. This way a probe image, taken in a random environment, can be recognized correctly.

3.1.3 Elastic Bunch Graph

Elastic Bunch Graph Matching (EBGM) [5], developed in 1997, is a biologically inspired algorithm utilized for recognition in the field of computer vision. This biological inspiration [19] can be derived by ,first, the visual features used are based on Gabor wavelets, which have been found to be a good model of early visual processing in the brain, more precisely simple cells in primary visual cortex and second, the matching algorithm itself is an algorithmic version of dynamic link matching (DLM) [20], which is a model of invariant object recognition in the brain.

Regarding the functionality of the algorithm, a set of database images is needed. Each image of the DB is represented with a graph of nodes and edges. The edges are labeled with the distance in between the nodes and the nodes are labeled with jets, as seen in Figure 3.3. A jet [5] describes a small patch of grey values in an image $I(\vec{x})$ around a pixel $\vec{x} = (x, y)$. It is based on a wavelet transform, defined as a convolution

$$J_j(\vec{x}) = \int I(\vec{x'}) y_j(\vec{x} - \vec{x'}) d^2\vec{x'}$$

with a family of Gabor Kernels y_j. Those Gabor Kernels are employed for different frequencies and orientations. This way, all kernels are being generated from one *mother* wavelet by dilation and orientation.

The number and position of the nodes selection is done customarily, but is based on covering key characteristics of the face. For example, as seen in Figure 3.4, four nodes can be selected, two of them being positioned in the center of the eyes and the other two in the edges of the mouth. For face recognition it is suggested that the nodes are

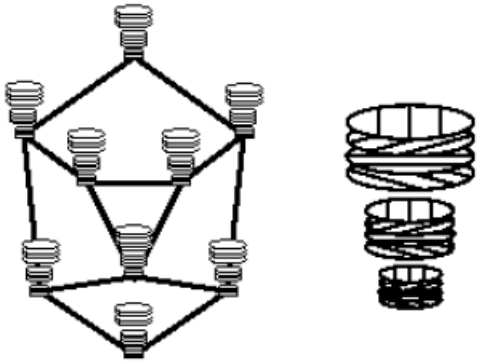

Figure 3.3: *An image graph with a jet seen on the left [5]*

placed in the inner face and not in the outline, as the recognition is based mainly on these features. The outline is used mainly for face detection.

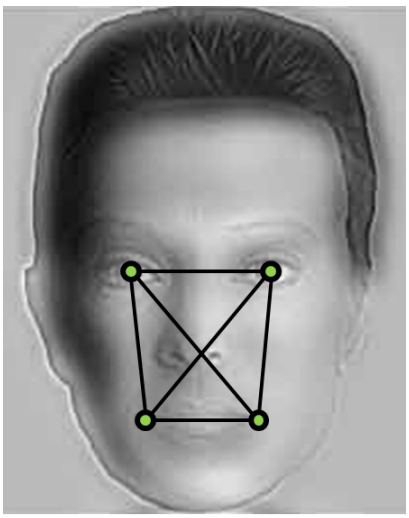

Figure 3.4: *Positioning four nodes and six edges applied in a face image.*

When nodes and edges, which are of the same number for all images, have been placed and labeled for every picture in the database, the average position of the nodes is being calculated. A face bunch graph can be then created, as seen in Figure 3.5, where the nodes are positioned in the average position and labeled with all images' jets. The set of jets in a node is called a bunch.

A probe image, to be recognized,is then inserted in the system. Initially the nodes are placed in the average position calculated before. The exact correct position of the nodes is then searched. To perform this, the jets for a region nxn around the given positions are calculated. A similarity function, utilizing values distance, is then used to define which jet position is more accurate. This way the right positions of the nodes are found and the jets are being calculated. The identification of the image is then taking

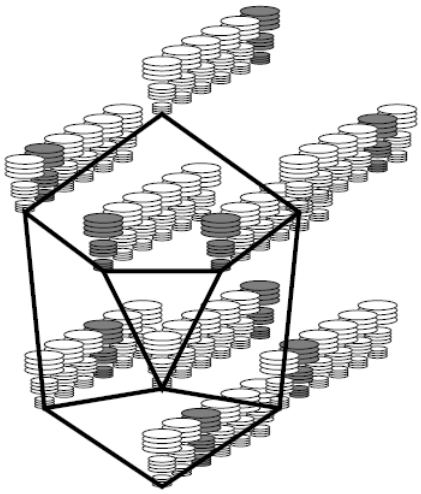

Figure 3.5: *A face bunch graph.*

place. The difference in the jets' values between the probe image and the database image is found and the smaller one is marked as a best match. Edges length can be also utilized in case of more than one close match.

The database for such a system depends on the number of images and the number of the nodes per image. Most researches select this number in the range 10-40. We can see, this way, that the database takes up small size. The calculations, though, needed to identify a probe image are of high number. This has to do, mainly, with finding the accurate position of the nodes, as the jets have to be calculated for every pixel in a frame nxn around each node. This increases significantly the calculations. Finally, the accuracy of the algorithm depends directly on the variety of images of a person in the database. Grids for different face rotations have to be included for the algorithm to be insensitive to rotations, as seen in Figure 3.6. Also pictures of different environmental conditions have to be included to effectively identify a probe image. This way the database size can increase remarkably. However, if all these specifications are followed the algorithm can achieve very high accuracy.

3.1.4 Local Binary Patterns Histograms

The Local Binary Patterns Histograms (LBPH), proposed in 2006 [6], is another algorithm used for face recognition, that is based on the local binary operator [21]. It is a very popular algorithm used widely because of its discriminative power and computation simplicity [22].

In more detail, a number k of gray-scaled images, sized NxM, is used as the database. First each image is split into sections. Usually the same size for splitting is used, in both width and height, resulting in mxm sections. Then for each image a histogram is calculated.

In each section the local binary operator is utilized. This operator, applied in images, compares a pixel to its eight neighbors. This comparison checks if the neighbor pixel is

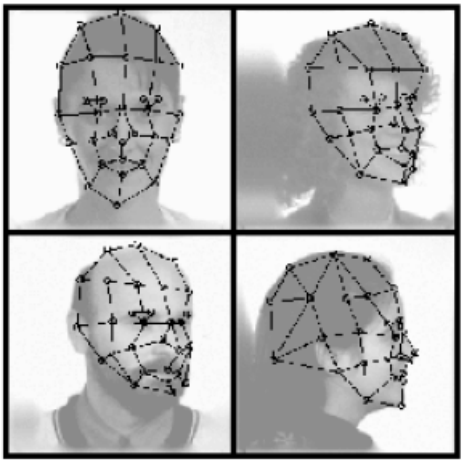

Figure 3.6: *Different grids for rotated faces.*

higher than the current pixel and if so, is labeled with '1' otherwise is labeled with '0'. Applying this to all 8 neighbors we end up with 8 binary values. Merging those values in a single number we have formed an 8-bit binary number that can be translated to a decimal number in the range 0-255. This operation can be observed in Figure 3.7. This process is done for every pixel in the section. Other variations of the algorithm, perform the operator in more neighbors or even in different distance from the center pixel, increasing in sometimes the accuracy.

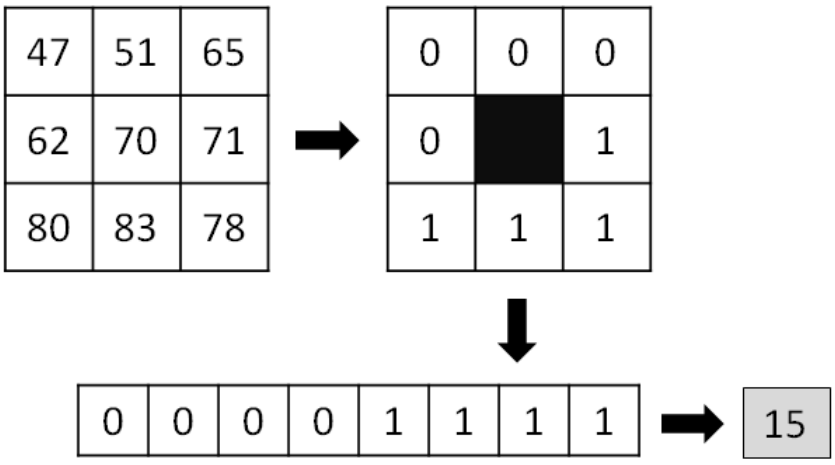

Figure 3.7: *LBP operator example.*

The histogram for the current section is then created by calculating how many times a value appears in this section. This way, the histogram for each section consists of 256

bins. This can be summed up to the following equation

$$H_i = \sum_{x,y} I\{LBP(J(x,y)) = i\}, i = 0, ..., 255$$

where H_i is the bin of value i, $J(x,y)$ is the (x,y) pixel of the image and I is an if operator returning '1' if the statement is true or '0' otherwise. When the histograms for all sections have been calculated a single histogram is create by unifying all the histograms for each section as seen in Figure 3.8. This final histogram will contain $256 * m * m$ bins and is defined as the feature vector of the image.

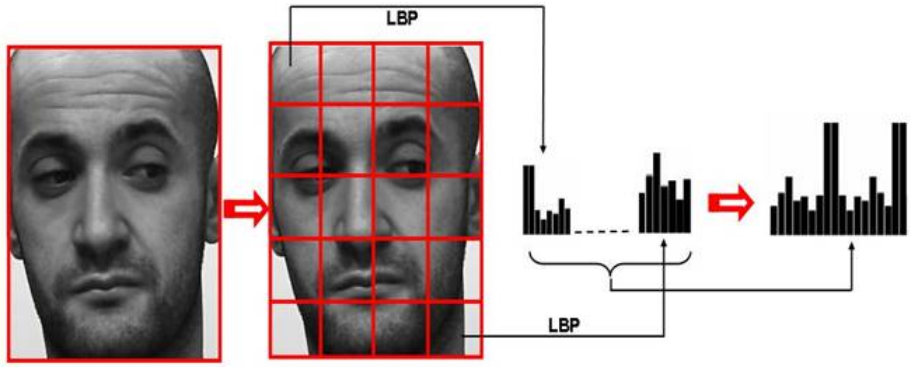

Figure 3.8: *Process of creating the final histogram [22]*

An image seeking recognition is now ready to enter our system. The same process is followed for this image to derive its own histogram. Once the histogram is retrieved, it is compared to the database's histograms and the closest match is identified, finishing the recognition process. Weights in different sections of the pictures can be added in this process. Sections that include important features (e.g. eyes, mouth) should be taken in more account when we try to find the closest match, and weights enable that.

An important extension of the algorithm is using classification. This way images of the same person can be added in the same class. When the comparison between the histograms of the database takes place, the distance from all the pictures in a class is averaged.

$$\Delta_n = \frac{\sum_{i=0}^{s} \delta_i}{s}, n = 0, ..., N$$

where δ_i is the difference between image's i histogram, s = size of class and N = number of classes. The probe image takes in account all pictures of a single person and not just one. This way the algorithm becomes more robust in different environmental conditions.

Two other extensions that are often seen, reduce the bins used in the histograms. The first one [23] , introduces the uniform patterns. Uniform patterns consist only of the values with up to 2 spatial transitions (bit wise 0/1 changes) in the binary vector. For example 00001100, 11110001 (2 transitions) are uniform patterns where 00110011 (4 transitions) and 01010111 (6 transitions) are not. Using only uniform patterns, the histogram bins reduce from 2^P to $P(P-1)+2$ for a P neighbor implementation. The second implementation [24], reduces the feature vector by considering the values retrieved

from the LBP operator and their complements as equal (e.g. 11110000 and 00001111). This way the bins in each histogram reduce from 2^P to $2^{P-1}+1$. Both implementations are very useful when the feature vectors need to be of small size and in addition robustness has been reported to environmental alterations in some cases.

Having said that, it is now obvious why this algorithm is preferred very often. The calculations are simple consisting mostly of simple pixel comparison. Also the database is small in size depending on number of sections per image and the number of images, $size = Images * Sections * 256$. Moreover, the LBP operator adds insensitivity to different illumination increasing the accuracy rate. Finally, through classification robustness is added to other conditions.

3.1.5 Scale Invariant Feature Transform

Scale Invariant Feature Transform (SIFT) [7] is an algorithm developed in 1999 for object recognition, but has been applied effectively for face recognition as well. Its main functionality is point matching between different views of a 3-D scene and view-based recognition. SIFT is invariant to translations, rotations and scaling transformations and robust to moderate perspective transformations and illumination variations. Experimentally, the SIFT descriptor has been proven to be very useful in practice for image matching and object recognition under real-world conditions [25].

The algorithm can be divided into two parts. In the first part, keypoints of an image are found. In the second part, feature vectors for this points are calculated. So, having a database of images in our system the keypoints of the images are found first.

The keypoints need to be invariant with respect to image translation, scaling, and rotation, and are minimally affected by noise and small distortions. The Gaussian kernels and next the difference of Gaussian function (DOG) in scale space are applied for this reason first, as they are the only smoothing kernels for scale space analysis [26]. For the first scale, each image is convolved with the Gaussian function in both axis.

$$G(x, y; k\sigma) = \frac{1}{2\pi k^2\sigma^2} e^{(-x^2+y^2)/(2k^2\sigma^2)}$$

Four convolutions for k, k^2, k^3, k^4 are performed in order to receive 3 DOGs. In this space maxima and minima are found. This is done by selecting pixels that are smaller or bigger than their 26 neighbors in the 3-D space. Those 26 neighbors are represented in Figure 3.9.

Next the image is scaled down. Billinear Interpolation is used in this stage to achieve correct scaling. Four convolutions with the Gaussian kernels are once more taking place but with k^2, k^3, k^4, k^5 this time. The minima and maxima are found in this scale DOG space as well. This process is repeated depending on how large the scale space needs to be. An overview of the process to receive the DOGs is shown in Figure 3.10.

After all this process a number of key points, that can be represented as different size blobs depending on the scale they correspond (Figure 3.11), has been found. An example image with keypoints is presented in Figure 3.12.The second part of the algorithm, for finding feature vectors for those key points, kicks in. For each keypoint, the magnitude M and orientation R for each pixel in a $16x16$ neighborhood (neighborhood size can be

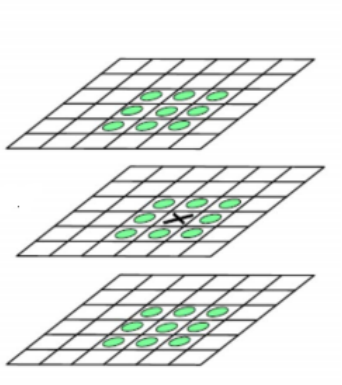

Figure 3.9: *Neighbors of a pixel in the 3-D space [27]*

extended depending on the design) are computed:

$$M_{ij} = \sqrt{(A_{ij} - A_{i+1,j})^2 + (A_{ij} - A_{i,j+1})^2}$$

$$R_{ij} = atan2(A_{ij} - A_{i+1,j}, A_{i,j+1} - A_{ij})$$

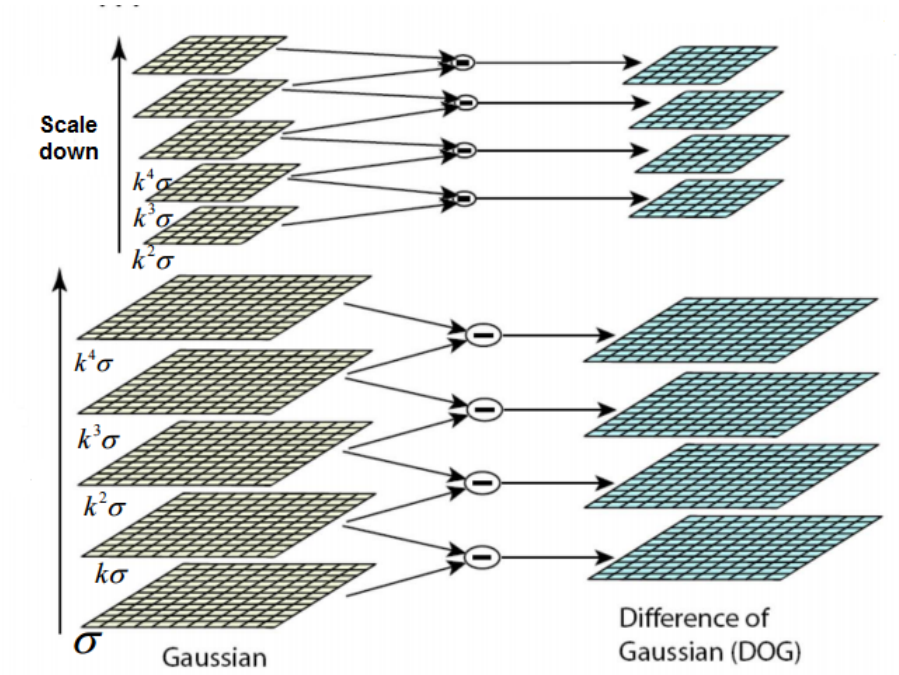

Figure 3.10: *Process to reach DOGs in keypoint exploration [27]*

By calculated magnitude and orientation, robustness in size and rotation is added. Moreover with utilizing neighbor pixels the algorithm can be invariant to illumination

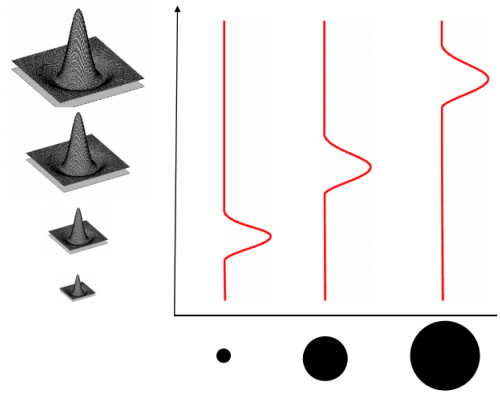

Figure 3.11: *Keypoints represented as different size bolbs depending on the scale [27]*

Figure 3.12: *Keypoints example of an image [25]*

changes as well. Moving on, the $16x16$ space is split into four $4x4$ sections. In each of these sections an 8 bin histogram is calculated. Each bin corresponds to one direction $0°, 45°, ..., 315°$ and the value of each bin is the addition of magnitudes for this orientation. The directions used, therefore the bins, may be of a higher number depending on how the algorithm is designed. The highest values direction is usually used to represent the orientation of the keypoint. This way, for each keypoint four 8-bin histograms are calculated (Figure 3.13). The unity of this histograms constitute the feature vector of this keypoint.

When a test image is inserted in the system the keypoints and feature vectors of the keypoints are calculated. The keypoints are then compared to the database images' keypoints and the one with the most matching keypoints is identified as the best match.

The SIFT algorithm introduces invariance in many different levels making it a very accurate recognition solution. A database for a system utilizing SIFT is small in size and depends on the keypoints per image, the orientations we want to include in the histograms and the images we will include. The computations, though, to recognize a test image are exceeding by far the calculations needed by other recognition algorithms. For extracting feature vectors, a series of convolutions, interpolations and summations are needed, making the algorithm really intensive. Moreover, this level of invariance is

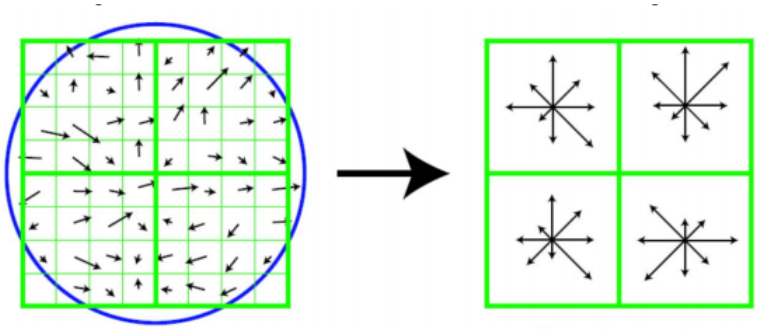

Figure 3.13: *Creating a histogram for a* $16x16$ *neighborhood using magnitude and orientation values [27]*

not needed in face recognition, as rotation and size variations are inside a certain range, contrary to object recognition. This way, the cost of the calculations is not reflected in the results' difference.

3.1.6 Algorithms Comparison

The most dominant algorithms for face recognition have been described in Sections 3.1.1-3.1.5. Their main functionality was explained briefly in order to understand the needs of each algorithm in calculations power, in database size, etc. Those algorithms have been implemented in the past and experimental results have been retrieved. In this section a comparison between them will be attempted in order to conclude to the algorithm that is most suitable for our implementation.

A comparison between the first two algorithms Eigenfaces and Fisherfaces is presented first, as those two algorithm have lots of common ground. In the first introduction of Fisherfaces [4], an extensive comparison between the two algorithms is reported. In all of the experiments the fisherfaces outperforms eigenfaces. The error rate difference is reaching even 30% in favor of fisherfaces, for complex image subsets. In Figure 3.14 the error rate for the algorithms is presented when the Yale face database is utilized, which contains variation in facial expression and lighting. It is obvious through this graph the difference between the two algorithms. Moreover, in [28], another comparison between the two algorithms is presented. In this research, the two algorithms are tested in different pose variations and the fisherfaces outperforms eigenfaces. Figure 3.15 shows the accuracy rate for different pose angle for the two algorithms. It has been clear that fisherfaces is a more accurate algorithm than eigenfaces. Also, the calculations needed for recognizing a face are of the same exact number for both algorithms, consisting of multiplying the probe image with the number of fisherfaces or eigenfaces. The database also is of the same size for both of them.

EBGM is examined next. The algorithm's accuracy was compared to different versions of eigenfaces and fisherfaces in the FERET Database which contains different subsets. The accuracy rate is checked for different ranks. In this point we should define rank as the number of best-matching images among which the correct answer is found. Rank is used oftenly to evaluate face recognition systems. In all of those experiments

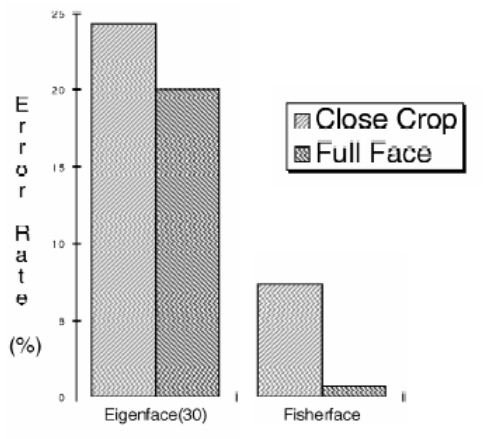

Figure 3.14: *Error rate of fisherfaces and eigenfaces in Yale Database [4]*

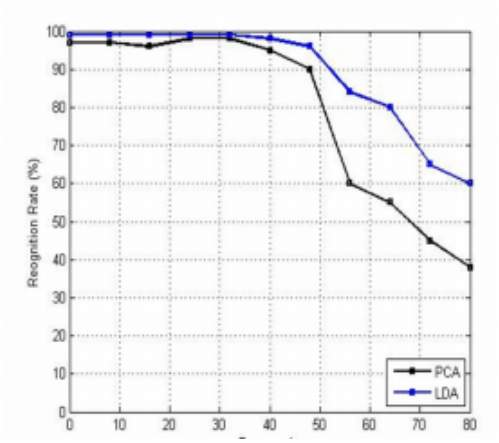

Figure 3.15: *Fisherface (LDA) and eigenface (PCA) accuracy rates for different pose angles [28]*

EBGM outperforms the other algorithms. In Figure 3.16 we can see the results for the fafb subset, images with alternative facial expression, and in Figure 3.17 the results for the dup2 subset, images taken in different time under different conditions. In both cases we observe the better accuracy of EBGM. Concerning the database needed for EBGM, is at the same size as the other compared algorithms. The drawback of EBGM is the heavy calculations needed for the probe image as described in Section 3.1.3 which may lead to a huge lateness.

Next, LBPH is investigated. We will concentrate on Ahonen's work [6], where an extensive comparison to the already proposed algorithms is being performed. LBPH is more accurate than the other algorithms in all of the experiments. Figure 3.18 we can see the performance of LBPH, with and without weights added (see Section 3.1.4), Eigenfaces, and EBGM optimal which is a better extension of EBGM. The experiments performed in the Feret database for the fafb (images taken with different expressions), the fafc (images taken with different illuminations), dup2 (images taken in different times

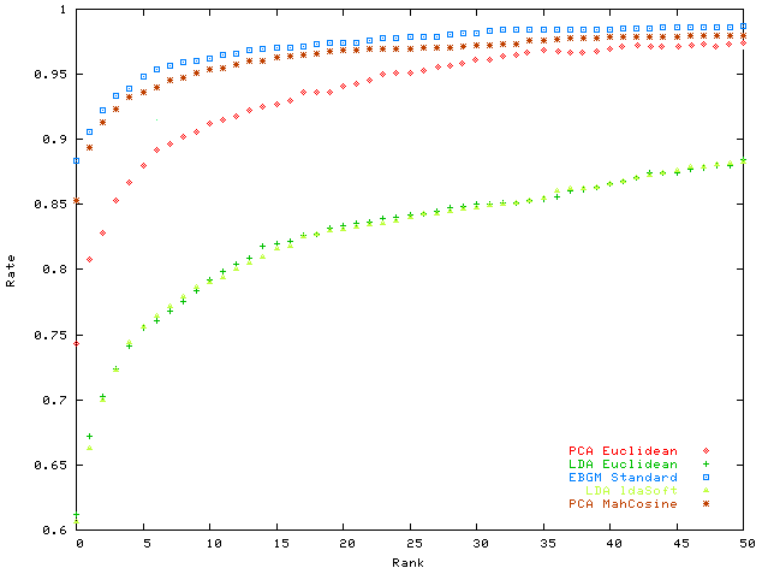

Figure 3.16: *EBGM, fisherfaces and eigenfaces comparison in Feret Database, fafb subset [29]*

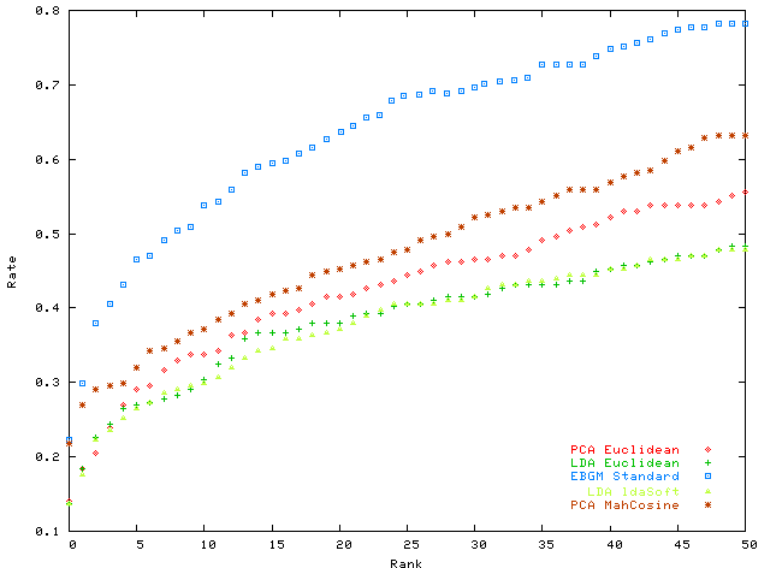

Figure 3.17: *EBGM, fisherfaces and eigenfaces comparison in Feret Database, dup2 subset [29]*

and conditions). For all subsets even the non weighted LBPH outperforms the other algorithms. Compared to EBGM, weight LBPH can report differences up to 40% for complex databases. Furthermore, the calculations needed for recognizing a test image are much less compared to EBGM, while the databases of LBPH systems remain in the same small size.

Finally, the SIFT algorithm is compared to the other models. As reported in Section 3.1.5 is a very accurate algorithm. Many researches, conducted through time, confirm

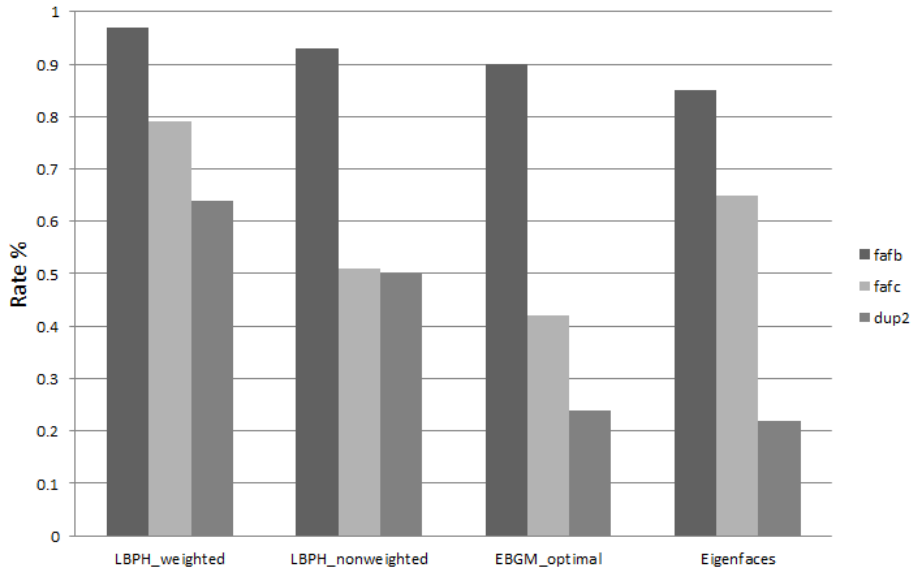

Figure 3.18: *LBPH, EBGM and Eigenfaces comparison in fafb,fafc and dup2 subsets [29]*

this[30] [31]. SIFT achieves higher accuracy rates outperforming all the other algorithms. In Figure 3.19 the accuracy rate for different image compression levels in an open universe environment are presented. SIFT results are better with a big difference compared to fisherfaces and eigenfaces methods. In comparison to LPBH, although the difference is much smaller with an average of 1.64%, SIFT is still returning better accuracy. Comparing the database size of SIFT utilizing systems to that of the other algorithms, similar size is reported. SIFT calculations though for recognizing a probe image exceed those of the other algorithms.

Below, Table 3.1 we give a brief estimate of the database size for the different models. In the features column we present the parameters for which the calculations are done. Those parameters are average approach and are used to have a general comparison. Also the image size is MxM and N images are used in the database. Also for every feature value we choose the 16 bits representation (2 bytes) as the range 0-65535 is adequate. In Table 3.2, an example database size is given for 100 images of size $100x100$. We observe that the sizes range from 49kb to 800kb which are very small values. As reported in Section 1.2, 100 images would be included in our database as well, making this example very useful. Although there are variations between the different algorithms, the size remains insignificant to affect directly our algorithm choice.

The calculations needed in average for each algorithm to extract the required features are presented as well. This is a very important part, as we do not want the calculation to be very complex in order to be translated to a low level logic effectively. Furthermore less intensive algorithms are also preferable to reduce the lateness boundary. As seen in Table 3.3 Eigenfaces, Fisherfaces and LBPH require the simplest and less calculcations of all algorithms containing matrix multiplications, additions and substractions. EBGM and SIFT need a large numbers of calculations which are also complex as convolutions are needed. SIFT specifically requires a huge amount of calculations that exceed by far

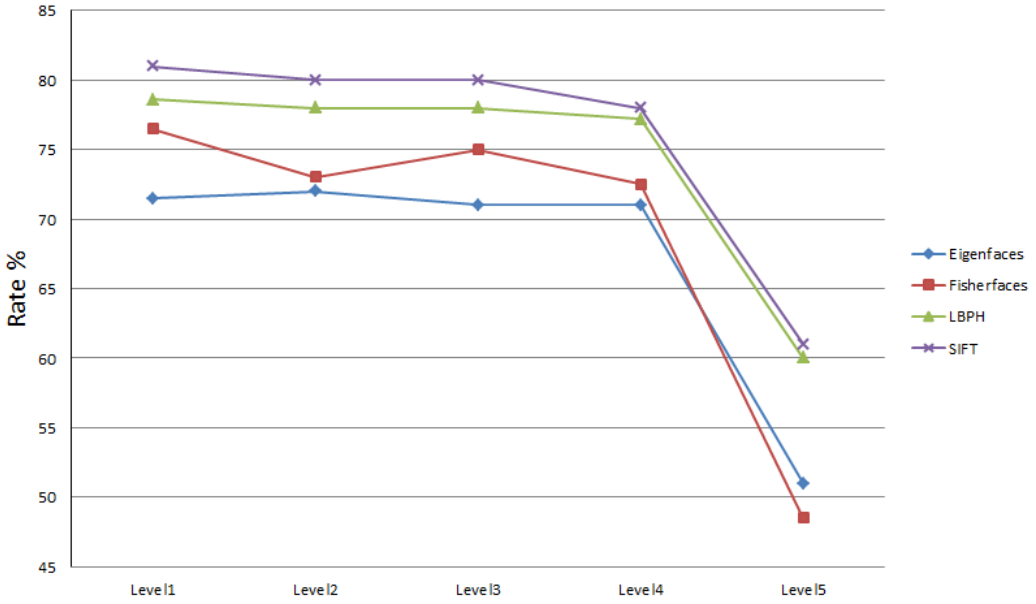

Figure 3.19: *Eigenfaces, Fisherfaces, LBPH and SIFT comparison for images with different compression level [30]*

Table 3.1: Algorithms' Database Size

Algorithm	Parameters	Database size (bytes)
Eigenfaces	*10 Eigenfaces used*	$10*MxM*2+N*10*2$
Fisherfaces	*10 Fisherfaces used*	$10*MxM*2+N*10*2$
EBGM	*10 Nodes used and 5 orientations,5 frequencies used in the Gabor Kernels*	$10*5*5*N*2$
LBPH	*16 sections used*	$16*256*N*2$
SIFT	*Average 20 keypoints, 4 neighbor sections used*	$20*4*8*N*2$

the simple calculations of Eigenfaces, Fisherfaces and LBPH.

The distance calculations for matching the probe image are not presented as they

Table 3.2: Database size for 100 images of size $100x100$

Algorithm	Database size
Eigenfaces	197kb
Fisherfaces	197kb
EBGM	49kb
LBPH	800kb
SIFT	125kb

Table 3.3: Calculations for extracting features from a $200x200$ probe image

Algorithm	Parameters	Calculations
Eigenfaces	*10 Eigenfaces*	10 MxM Matrix Multiplications
Fisherfaces	*10 Fisherfaces*	10 MxM Matrix Multiplications
EBGM	*10 Nodes, 5 orientations and 5 frequencies in the Gabor Kernels and a frame 3x3 for node positioning*	$10 * 5 * 5 * 9$ MxM Convolutions and $3 * 3 * 10 * MxM$ Subtractions
LBPH	*8 neighbors used*	$MxM * 8$ Subtractions and MxM Additions
SIFT	*3 x.5-scales used, 4 4x4 neighbor sections, 20 keypoints*	16 MxM Convolutions, 9 MxM Subtractions, $\frac{3*MxM}{4}$ Interpolations , 20*16*16 Tangent calculations, 2*20*16*16 additions

are the almost identical for every algorithm. Their only difference is relevant to the number of features that are extracted from each algorithm, which depend mainly on the parameters used in each implementation. With the utilization of similar size parameters, the number of features remain almost the same for each model.

Depending on those facts, the proper algorithm should be chosen for our system implementation. Regarding accuracy, SIFT returns the highest rates followed closely by LBPH. Fisherface and EBGM are below LBPH and Eigenface comes last with a very low accuracy rate, making it an insufficient algorithm for our system. Also SIFT and LBPH support light invariance which is a very important aspect. For this reason, alongside with the high accuracy rate, those two algorithms are the main candidates for our system. The high and complex calculations, needed by SIFT to be applied on the probe image, though, set a huge drawback for this algorithm. The average difference in accuracy between LBPH and SIFT of 2% , as stated before, is not justified by the huge difference in calculations needed. As mentioned in Section 3.1.5, this difference could be reflected in a different subject implementation e.g. object-recognition, but not in face recognition.

In conclusion, the above comparison results presented, lead to the decision to utilize Local Binary Patterns Histogram (LBPH) for our system. Through this algorithm comparison, one can comprehend effectively and verify the proneness of LBPH for the current project implementation.

3.2 Local Binary Patterns Histograms Implementations

LBPH as mentioned before, is a very popular algorithm. Numerous systems utilize it for their own purposes. In the previous section, we saw how the algorithm works. Although the calculations needed by the algorithm are not many compared to other algorithms, they still require a noticeable execution time. These number can also increase depending on the size of the image and the parameters the algorithm is using (e.g. number of

sectors). Solutions that try to reduce the execution time of LBPH, while maintaining its accuracy have been investigated. As LBPH was introduced in 2006 [6], the first researches on this subject lay only a few years back, denoting its novelty.

In 2010, an implementation of Local Binary Patters was implemented using SIMD (Single Instruction Multiple Data) Instructions of CPU [9], in order to achieve acceleration of the algorithm. This implementation aimed on object detection and focused on the local binary operator only,as no histogram creation exists. The advantage of this implementation is that no extra hardware is needed as an SSE supporting CPU is adequate for it. Except from the SIMD instruction, in this work the number of memory accesses are minimized. This is done by preprocessing the image. Through interleaved convolution the needed data are extracted fast and are placed in the same block for each feature examined, minimizing the number of accesses needed. After the preprocessing of the image the SIMD is utilized for the LBP operator on the data. Figure 3.20 presents an example of the LBP operation in a set of data. The SIMD enables these operations presented to be done in parallel. This implementation was compared to the implementation with no optimizations that is referred as 'Plain C'. The results of the comparison can be seen in Table 3.4. A large difference in frames per second process can be viewed.

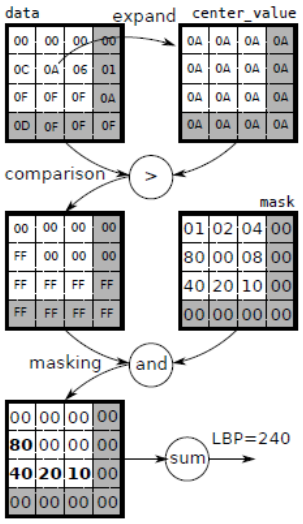

Figure 3.20: *Block diagram of SIMD LBP evaluation [9]*

Another implementation of LBPH, applied specifically for face recognition, was introduced in 2012 by Tek et. al [8]. In this work, CUDA is used to implement the algorithm in GPU, in order to accelerate it. Two methods are approached to achieve this. First, each thread block is responsible for retrieving the histogram for each region of the image. During this process, each wrap calculates a part of the region and when all wraps are finished the final histogram is constructed. A representation of this can be viewed in Figure 3.21. Second, the matching process is parallelized. This way, each block, and each wrap consequently, can perform the distance calculations between the feature vectors in parallel, accelerating the process. In Table 3.5, the results of the GPU are compared to the CPU implementation. Different region sizes, feature lengths and

Table 3.4: Object detection in *frames per second* for two different target error rate a classifiers.

$a = 0.1$	560x240px	720x576px	1280 x 720px
SIMD	61	22	10
PlainC	8	3.4	1.4
$a = 0.2$	560x240px	720x576px	1280 x 720px
SIMD	87	28	13
PlainC	12	4.5	1.9

Table 3.5: Feature extraction times of the GPU and CPU implementations for various cases using a single stream [8]

LBP Type	Region Size	Feature Length	GPU [ms]	CPU [ms]
$(8,2)^{u2}$	11x13	8496	0.17	4.25
$(8,2)^{u2}$	18x21	2891	0.17	3.85
$(8,2)^{u2}$	26x30	1475	0.17	4.01
$(16,2)^{u2}$	18x21	11907	0.18	7.45
$(16,2)^{u2}$	26x30	6075	0.18	7.86

LPB types are examined. We should mention that the format (e.g. $(8,2)^{u}2$) for the LBP types reports the pixel neighbors used, the distance from the centre pixel and also if an extension of LBP is used, $u2$ in this case corresponding to uniform extension. We can observe that a noticeable speedup is achieved using the implementation proposed in GPUs. The difference between the two execution times tends to become larger, when the data processed are increasing.

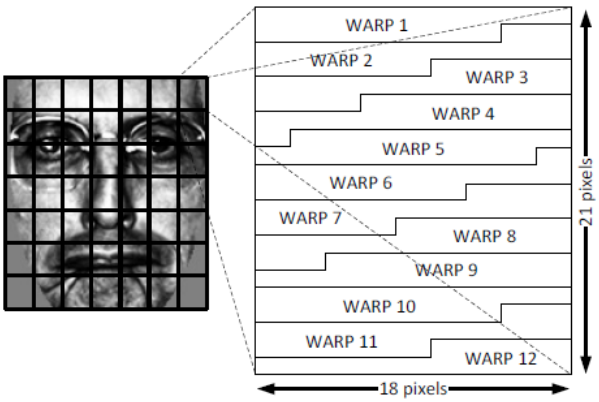

Figure 3.21: *Arrangement of the thread blocks for a sample image divided to 7 7 regions. Each warp in a thread block constructs a per-warp histogram for a 32 pixel area in the corresponding region as shown on the right side of the figure. [8]*

An FPGA implementation of LBPH was introduced in 2012 [32], concentrating in

Table 3.6: Detection results. Correct detections (CD), false detections (FD), missed detections (MD), accuracy = CD/(CD+FD) [32]

	CD	FD	MD	Accuracy
Training dataset	613	0	0	100%
Test dataset	296	9	11	97%
Annotated video	496	99	140	83%

human detection. A video-stream of resolution 640x480@60fps was used as an input and the system designed managed to detect humans in real time. Parallelizing many computations in the FPGA leads to a better execution time. The uniform and non-redundant extensions of LBPH were utilized minimizing the feature vector size. The SVM classifier was also used for effective classification. In Figure 3.22 the system overview can be viewed. All the modules presented where designed using VHDL and Verilog. The implemented system was tested for accuracy. The results were very impressive and can be seen in Table 3.6. Regarding the execution time, real time execution is reported as achieved, but unfortunately no timing results are presented. Although this work approaches our project definition, only small feature vectors are used and this work is limited to detection.

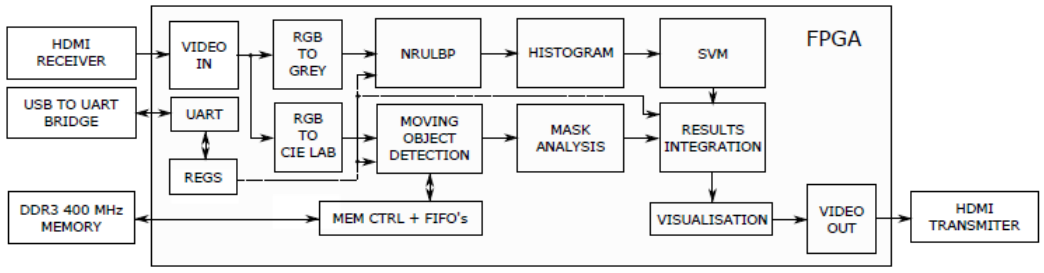

Figure 3.22: *FPGA based system overview. [32]*

In conclusion, through all the works presented, LBPH is returning good experimental results. The researches carried out indicate that the algorithm can attain low execution time if the right resources are utilized. GPUs , FPGAs or even CPUs can lead to this result. Parallelization of the algorithm, or parts of it, is though needed in order to achieve that. Different approaches have been presented in this section leading to the same goal through different paths.

3.3 Conclusion

Through this chapter, a literature review on the field of facial recognition was performed. In Section 3.1 the most popular and cutting-edge facial recognition algorithms were presented. The main points discussed were the functionality of each model, its efficiency and its complexity. An algorithm comparison was also presented, in order to identify

the most proper one for our implementation. Through this process the LBPH model was selected. In Section 3.2, different modern implementations of LBPH were presented. Those implementations utilize cutting-edge technology in order to accelerate the algorithm and optimize its performance. Examining those implementations, the novelty of our system can be pointed out, as it is the first time the LBPH algorithm will be designed in an FPGA for face recognition purposes.

4 Algorithm Development

In the current chapter, the process of developing the algorithm will be presented. The Local Binary Patterns Histograms model is utilized for our system. The model structure and functionality has been described in Section 3.1.4. Also in Section 3.1.6 a comparison between other models has been presented, justifying our choice to utilize the LBPH model.

The developing of the model includes writing the basis code for the complete algorithm, choosing the proper parameters for maximum efficiency and testing the performance. The parameters that are going to be examined are: the number of image regions used in LBPH, the weight of the different regions, the distance measure that is going to be utilized to compare the histograms and finally the use or not of classification. The utility and effect of these parameters can be observed in Section 3.1.4.

Matlab will be utilized for prototyping the code. For testing the performance of the algorithm, the Unconstrained Facial Images (UFI) [33] database will be utilized. A more detailed description of this database will be provided later on in the document, as in the current section this shall not be our concern. The database will consist of 100 images and 100 images will be used for testing.

During the development of the algorithm our focus will be centered in the first rank accuracy results, as those are depended solely from the parameters used. The performance should not be our main concern at this point, as this will be the main aim during the design and implementation of the model on the FPGA-based SoC will focus on the performance aspect and apart from that the parameters effect it in a small percentage.

In the following sections the procedure of choosing the proper parameters is presented in detail.

4.1 Number of Regions

In LBPH the image is divided into regions, as described in Section 3.1.4. For each region, a unique histogram is formed from the region pixels' LBP values. The complete image histogram is created by merging all the regions' histograms. Having said that, is obvious that the more the regions, the larger the size of the histogram and the database. However, the number of regions is affecting the accuracy of the recognition as well.The research study presented in [34] shows that the dependency between the region number and the accuracy is of concave and monotonously increasing form.

In the current work the effect of different region numbers is examined too. It should be mentioned that as the input images, as defined in Section 1.2, are of size 200x200, the division by the region number should return a zero remainder. Once the prototype code has been developed (the chi-square distance measure and classification is used at this point to compare the histograms), different region numbers have been tested. The graph

Table 4.1: Distance between the best match and the second best match distance.

Region Numbers	Distance
4x4 (16)	23.88
5x5 (25)	24.62
8x8 (64)	25.12
10x10 (100)	23.16
10x20 (200)	21.04

below, Figure 4.1 shows the accuracy for the different region numbers. The distance between the best match distance and the second best match distance is also examined to observe the robustness of the model. Those distances are presented in Table 4.1. Finally, the size of the histogram for the different regions is presented in Figure 4.2.

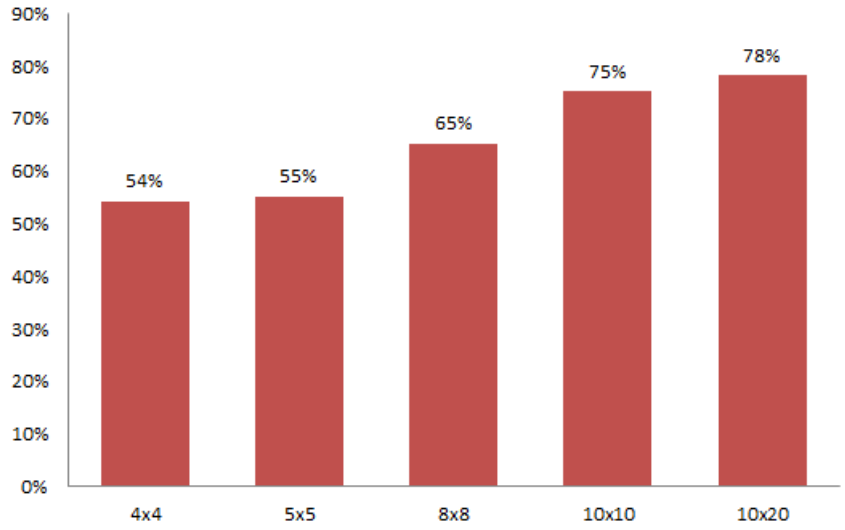

Figure 4.1: *Accuracy results for different region numbers. (X axis x Y axis)*

The above presented results provide us a complete overview of the affects the different region numbers have. The logarithmic-like form, mentioned before, between the accuracy and the region numbers can be confirmed by the results in Figure 4.1. The accuracy is increasing with a bigger rate in the lower region numbers and less as the numbers increase. Between the 10x10 and the 10x20 region numbers only a 3% difference is reported. Those two cases are the best candidates for our final model, as they provide an acceptable accuracy rate.

The size of the each image histogram size for 10x10 and 10x20 regions is 50kb and 100kb respectively. Having in mind that the database will consist of numerous histograms, the 10x20 regions case will always lead in a double size database, which might be prohibitive in larger databases. Moreover the calculations needed to compare a test image to the database will be doubled for the second case. This, double increase in both the database size and the calculations is excessive for returning an only 3% improved

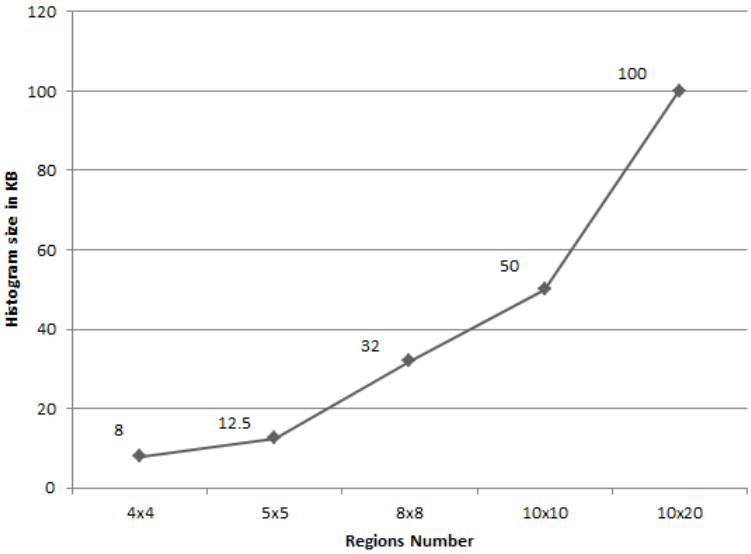

Figure 4.2: *Image Histogram size for different region numbers.*

accuracy. Also, the better robustness reported, through the results in Table 4.1, for the 10x10 region size offsets in a small scale the lower accuracy percentage.

In conclusion, the 10x10 regions number (100 regions) is selected for our model. The ratio of accuracy and robustness to database size and calculation operations was the main criterion to lead to this decision. Through this selection a high accuracy is obtained for our model.

4.2 Weights of Regions

The different regions of an image form a kind of an entity, as they include a unique histogram. It is clear that some parts of an image (face) help to greater extent in recognizing a face. Thus some regions play a more important role in face recognition than some others, which are less significant.

In the current section the effect of region weights will be examined (the same code as in Section 4.1 was used with 10x10 regions). The first set of weights utilized is derived by [6], where an efficient weight set for a 7x7 regions image is presented. Because of the different region number in the current model, 10x10, the weight set modified to match the different regions dimensions. This modification, that leads to the first set of weights utilized, can be observed in Figure 4.3.

The second set of weight was derived by a set of experiments based on the first set of weights. During these experiments the variance between the different regions was examined to identify important regions that may vary the first set. Based on this variance, a sequence of modifications on the first set were applied and the accuracy was tested. The final, most accurate, set of weights derived is presented in Figure 4.4. The derived weights were initially 0.5, 1, 2, 4, but in order to avoid floating point operations the weights utilized were doubled.

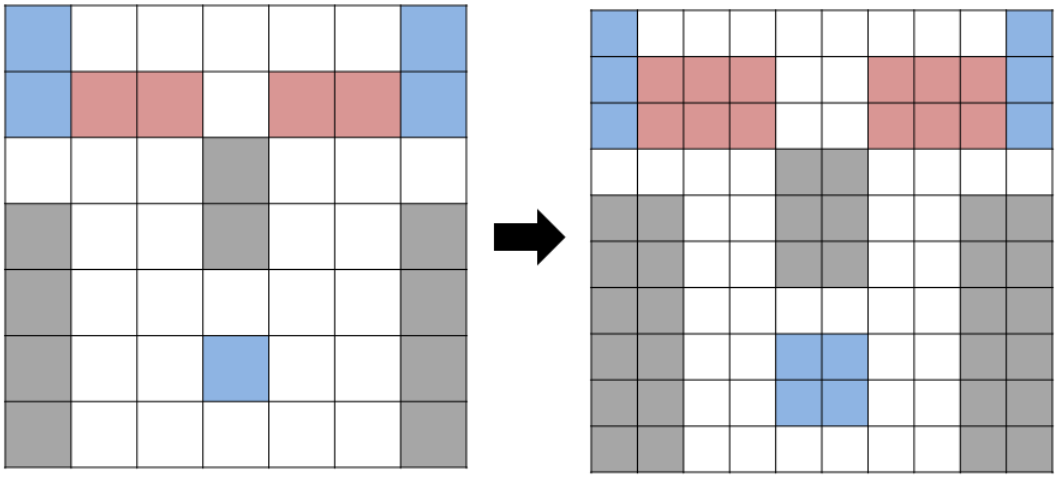

Figure 4.3: *7x7 regions weights [6] modified for 10x10 regions (red squares indicate region weight 4.0, blue 2.0, white 1.0 and grey 0.0)*

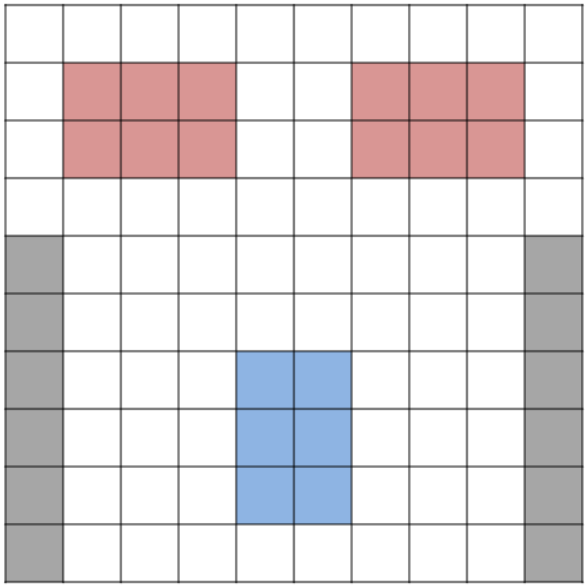

Figure 4.4: *Proposed region weights (red squares indicate region weight 8.0, blue 4.0, white 2.0 and grey 1.0)*

In Figure 4.5 a comparison between the different region weights is presented. The proposed set of weights (second set) returns an 1% higher accuracy than the Ahonen weights (first set). This improvement does not burden the performance of the algorithm and therefore will be the set utilized in our model. It is observed that the overall accuracy has been increased by 3% with the addition of proper weights, leading to an overall accuracy of 78%. This rate for first rank indicates a high-accuracy model for face

recognition.

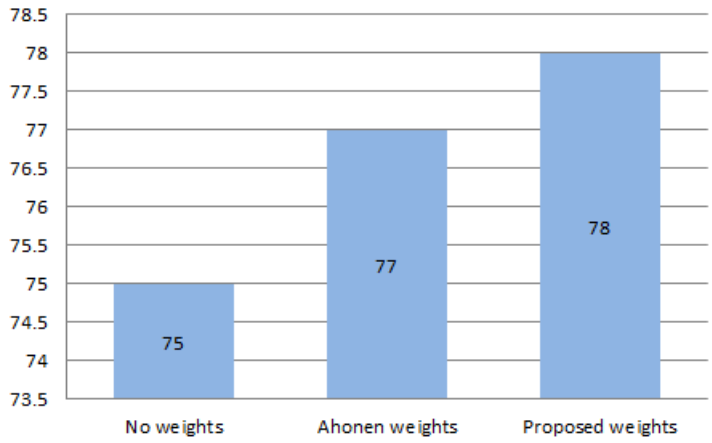

Figure 4.5: *Region weights comparison (No weights, Ahonen weights [6], Proposed weights)*

4.3 Distance Measures

Up to this point, the experiments conducted for the development algorithm utilized the chi-square statistic for the distance measure. In this section three more popular distance calculation models will be examined to identify the most proper one for our implementation. The four, in total, distance measures examined are Chi-Square, Euclidean Distance, Manhattan Distance, Log-likelihood. The distances for each one are derived by the following mathematical models:

- Chi-Squrare statistic [35]:
$$\Delta_{xy} = \sqrt{\frac{(x-y)^2}{x+y}}$$
- Euclidean Distance:
$$\Delta_{xy} = \sqrt{(x-y)^2}$$
- Manhattan Distance [36]:
$$\Delta_{xy} = |x-y|$$
- Log-likelihood :
$$\Delta_{xy} = -x \log y$$

The mathematical models presented have a wide range of utilization from simple distance calculations to statistical methods. Moreover, their utilization has been reported in the past for histogram comparisons [37]. Those four models are considered as the distance measures candidates for our model. The algorithm developed so far (10x10 regions, proposed region weights) was used to evaluate the four models. The same test experiments was conducted four times with a different model utilization each time for

calculating the distance between the test histogram and databases histograms. The accuracy results obtained are presented in Figure 4.6.

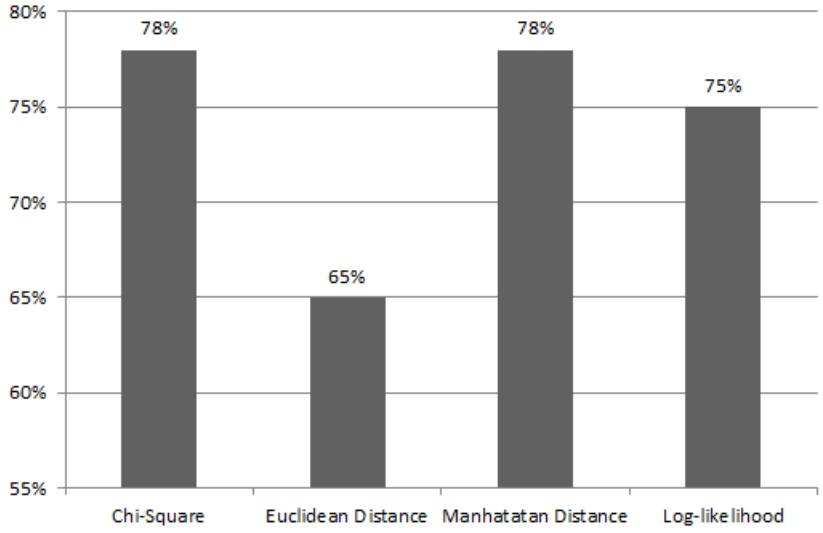

Figure 4.6: *Accuracy of different distance calculation models*

Through the graph in Figure 4.6, we can notice that the two most accurate models are the Chi-Square and the Manhattan Distance with the same accuracy, 78%. The decision of the distance measure for our implementation has to be made between those two models. Although the same accuracy is returned, the Manhattan Distance consists of much simpler operations. Even if the square root is removed from the chi-square, as the distance differences will remain the same, a division operator is included in the model. The implementation, as described before, will be accelerated using an FGPA, which requires many clock cycles to perform the division operator and thus should be avoided. In addition, the division in the model may lead to floating points, making this way the whole design more complex. On the other hand, the Manhattan Distance is a simple model requiring only an absolute value operator, which can be implemented easily on the FPGA.

To sum up, the Manhattan distance is utilized to calculate the distance between the histogram derived from the test image and the database histograms. This model returns high accuracy recognition and is simple to implement on our system.

4.4 Classification

The prototyping of the code was based on classification utilization. In this section, classification will be examined. As classification, we define the procedure of creating classes of common database faces, in order to make the recognition more accurate.

The purpose of classification in recognition is to take in account all similar faces before making a matching decision. This way, the recognition becomes robust to extreme variations in a face image and the matching process is being normalized. Moreover, the

faces included in a class are very important. If a class includes images that cover a variety of conditions (expressions, poses, etc.), the recognition of a similar face is more easily performed correctly. For example, a class containing a person's images while he is laughing or with closed eyes is more appropriate than a class with different images of the same pose.

As the face recognition process aims on recognizing people, the classes are formed including the same person's face images. As described in Section 1.2, in the current project, the database will include 100 faces of 20 different people, 5 images per person. It is clear that the classification process will create 20 classes each one containing 5 face images.

Through classification, the matching of a test face image takes place in the class level and not in the separate face images level, as seen in Figure 4.7. However, this matching process may vary and it is going to be examined in order to choose the most proper one.

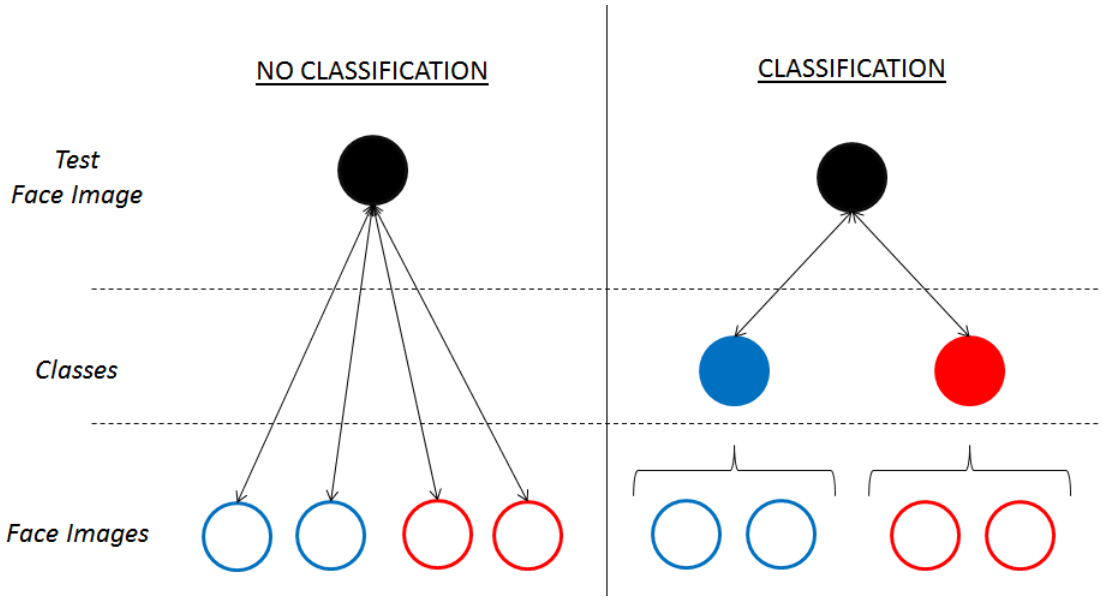

Figure 4.7: *Matching process with and without classification*

As described in the previous section, the Manhattan Distance is used as the comparison mean. Through this model and with the utilization of classes, we can define two cases for the matching process that are going to be investigated. Those two cases, differ on the distance calculation between the test image and the class and are defined by the following models:

- Case 1:

$$\Delta_{tc} = \frac{\sum_{i=1}^{N} |t - im_i|}{N}$$

- Case 2:

$$\Delta_{tc} = |t - \frac{\sum_{i=1}^{N} im_i}{N}|$$

,where t is the test image histogram, im_i are the face images' histograms of the class and N is the size of the class. In the first case the Manhattan Distance is calculated for every image histogram of the class and then the average is found. In the second case, the average histogram of the class is found and then the Manhattan Distance is calculated. The second case, is considered as the average histogram of the class can be calculated before hand and be included to the database. This way the database is reduced to the number of classes (reduction of 5 times in this project).

The matching process is performed by matching the test image to the closest distance class, thus to the closest person. The two cases alongside with the no classification case, where the closest distance test image is indicated as match, where tested. The accuracy results can be seen in Figure 4.8

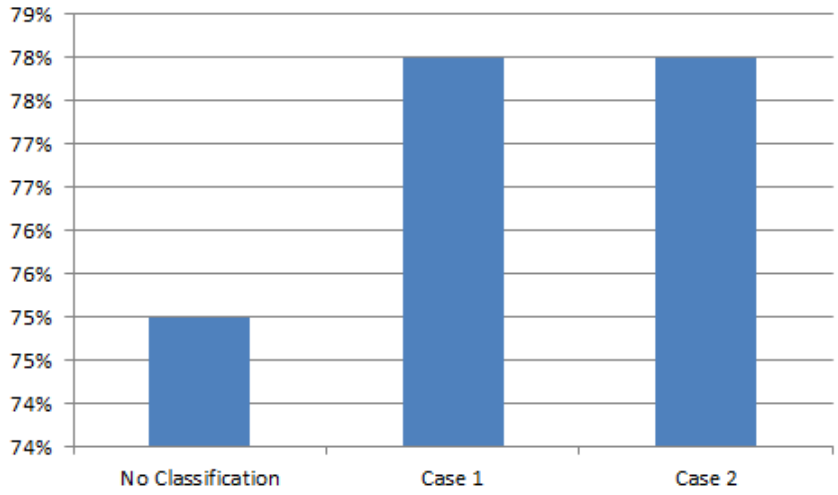

Figure 4.8: *Classification cases accuracy results*

As seen in the graph, the benefits of classification are observed. The robustness added to the algorithm leads to better accuracy results. The classification enhanced the accuracy by 3%. Also, we observe no difference in the accuracy between the two matching cases. This makes the second case as the most proper matching case. As described before, in the second case, the database size is reduced to the number of classes, which is crucial for both the memory needed by the system and also the comparison calculations needed and as there is no significant drawback on the accuracy this is the matching model that will be utilized for our system.

Utilizing this model, the formation of the database of our system will be done by finding the average histogram for each class. Through this process 20 histograms will be calculated and imported to the memory of the system.Then, the test image's histogram distance from the classes will be calculated according to the case 2 mathematical model described above. However, the average histogram calculated may contain floating values. This is something we want to avoid for our system, in order to decrease its complexity. To overcome this issue, the sum of the class images' histograms is calculated instead and imported to the system. The distance calculation, which has to remain accurate, is then

performed by the following model:

$$\Delta_{tc} = |N * t - \sum_{i=1}^{N} im_i|$$

Through classification, the system is benefiting on two different aspects. First, the algorithm becomes more robust on face images conditions and therefore more accurate. Second, the database sice, through the matching model, is reduced to the number of classes, 5 times smaller database in the current system. Those conclusions justify our decision to use classification in the first place and furthermore to utilize the second matching case, presented, for our system.

4.5 Conclusion

The complete development of the algorithm has been presented in this chapter. This process is mandatory before proceeding to the system implementation. The proper parameters and models for the system need to be examined, prior to the implementation, in order to ensure the efficiency of our system. The finalized developed prototype algorithm will set the basis and guidelines for the implementation.

The key aspects investigated included parameters of the LBPH algorithm and included models. In Section 4.1, the number of regions of LBPH was decided, assuring high accuracy and relatively small database. Section 4.2 describes the way the proper region weights were chosen to further increase accuracy. Different distance measures, needed for the histogram matching process, were presented in Section 4.3, and the most appropriate one, Manhattan Distance model, was selected. Finally, the utilization of classification and its benefits were described in Section 4.4.

In conclusion, through a set of experiments, the algorithm for our system was developed following the most appropriate path. This process' goal was to maximize the efficiency of the algorithm, always with respect to the project specifications. This goal is confirmed by the final 78% first rank accuracy achieved and also by ensuring that the model complexity is not burdened.

5 Implementation

The process of the system implementation will be described in this chapter. This implementation is based on the developed algorithm, Chapter 4, and with respect to the project specifications, described in Section 1.2. The platform utilized is the Topic Development Kit (TDK), presented in Section 2.3, and the goal is to utilize successfully its resources, to achieve an efficient design. In the following sections, 5.1 - 5.4, the whole procedure of implementing the final system will be described extensivelly.

5.1 Design Approach

An investigation on the general implementation approach is initially required. The purpose of this investigation is to examine which resources should be utilized, to achieve our goals and in which way. We would like to repeat, in this point, that the main goal of this implementation is to accelerate the algorithm, in order to achieve the real-time target of 0.16 seconds. The other goals of this project, such as high-accuracy, have been guaranteed through the algorithm development, leaving the execution time, as the only concern for the current design implementation.

As the platform utilized is the TDK, the prototype Matlab code has to be written in C, in order to examine its operation on the ARM processor. The developed code is then profiled to have an insight on its performance. The parts of the code examined were the Feature Extraction, utilizing LBPH, and the Histograms Comparison, utilizing Manhattan Distance, which were grouped into two separate functions, to have a clear overview. The input and output operations were not examined, as this is depending on the different systems' needs. The total execution time for those two functions, on the bare-metal ARM, was summed to 1414.397 milliseconds (ms). The feature extraction required 514.799 ms and the histograms comparison 899.598 ms. The ratio between those two function can be seen in Figure 5.1.

The execution times are far from the goal of 16ms, as seen in Figure 5.2. The processing units of the TDK are two ARM processors and an FPGA. Even if we managed to perfectly parallelize the code to run in the two ARM processors concurrently, the execution time would be reduced to half. This reduction is not enough to satisfy the real time goal. Therefore the utilization of the FPGA is mandatory for implementing both parts of the algorithm.

Two IP cores (Intellectual Property cores) need to be designed for the FPGA. One for the LBPH, feature extraction, and one for the Manhattan Distance, histograms comparison. The reason those two are separated and not designed in one IP, is for maintaining a less complex design. Furthermore, the LBPH part would be the same in every future system, while the Manhattan distance may differ depending on the database size. Therefore, taking in account future systems development, is another reason for this

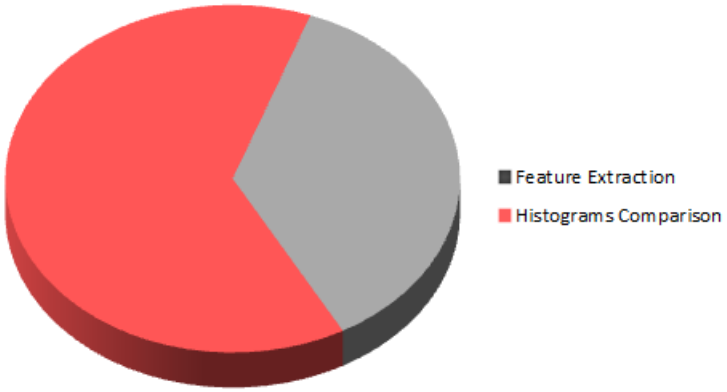

Figure 5.1: *Feature Extraction, Histograms Comparison ration*

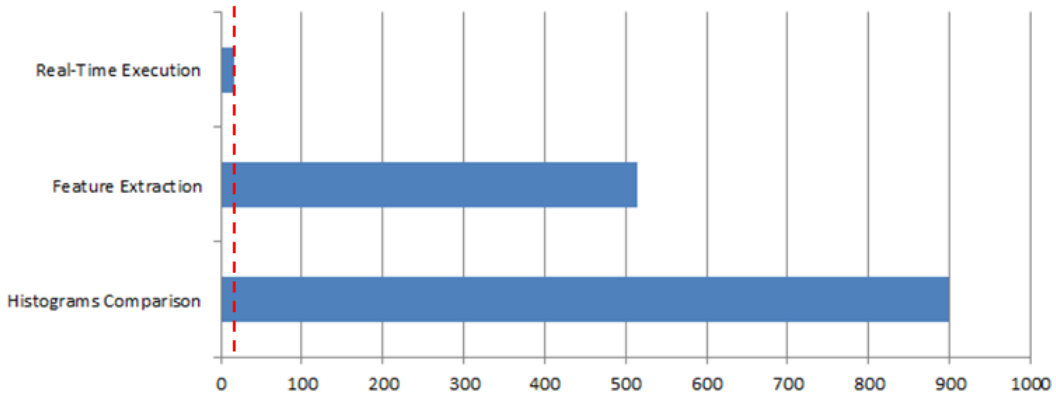

Figure 5.2: *Real-Time execution compared to Feature Extraction and Histograms Comparison time*

design.

Another aspect, regarding the system's universality, is the data transfers of the FPGA. For both input and output, the data transfers, will be done through the DDR of the system. This way, a wide range of data importing ways are supported, as saving data to DDR is a simple task. The same applies to the output, as it will be stored in the DDR. Through that, presenting the result in any wanted way (output to monitor, to console etc.) is made easy.

The ARM processors will be utilized in the system for sub-tasks. Those tasks include the data transfers from and to the DDR, managing FPGA interrupt signals and finally initializing the memory access for the FPGA, through DMAs, which will be described in more detail later.

The above described analysis, lead to the final system design. This overview of the system can be seen in Figure 5.3. The main aspects that lead to this design are, first, the system's performance, that needs to meet the real-time goal, second, the system's

simplicity and finally the availability for development differentiations.

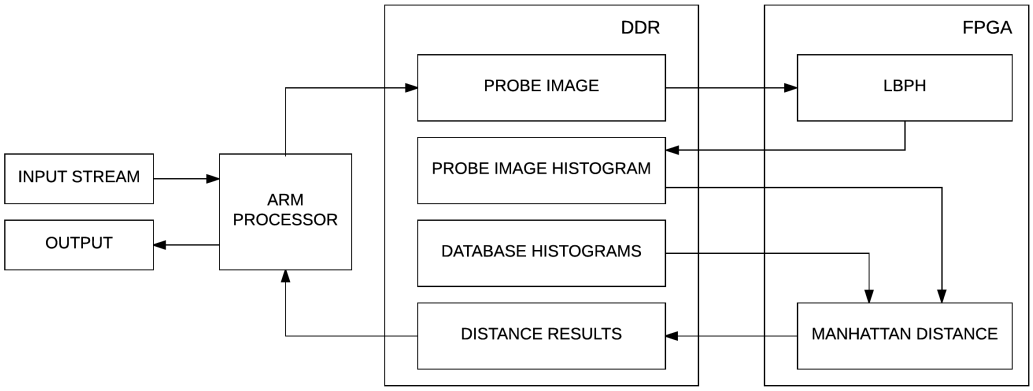

Figure 5.3: *System Overview [38]*

5.2 Feature Extraction IP core

The feature extraction IP will be designed for the FPGA first. To develop the required VHDL code, the Vivado and Vivado HLS tools will be utilized. The purpose of this design is to create an IP to function as the developed algorithm, but perform much faster than the original C code.

Initially, the baseline code for the core is written in VHDL. Many aspects have been taken into consideration during this process. First, The input and output of the block are designed based on the Advanced eXtensible Interface 4 (AXI4) protocol [39] supported by Kintex-7. In specific, the AXI4-stream interface is utilized as the data will be received in a stream from the DDR to increase the performance. Through AXI, high speed transactions are enabled, which is mandatory for our system.

Second, block RAMs should be used for storing the input and the temporary output data. This is necessary as the computations performed in the algorithm do not fetch the pixels sequentially, but in a random-like way. For this reason, the whole set of pixels, 40000 ($200x200$), needs to be available during the whole process. The same applies for the temporary output data, as the LBP values computed, match different, out of order, bins on the histogram. Furthermore, the BRAM values of the temporary output need to reset every time the core is utilized, to ensure correct functionality. The reset of the input BRAM values is not needed, as they are replaced by the new input values every time,in contrast to the temporary output values that are just increased and not replaced. In order to avoid the reset of the whole BRAM IP, which is much complex and time inefficient, the BRAM values will be reset internally, through the design, every time the core is utilized.

The main parts of the IP core design are the following:

- Input and Output ports using the AXI4-Stream interface are designed. The input

Table 5.1: Baseline design performance in clock cycles

Process	Latency	Iteration Latency	Trip Count
1	40000	1	40000
2	25600	1	25600
3	880000	22	40000
4	51200	2	25600
Total	996800		

should consist of the 40000 pixels of the input face and the output of the 25600 bins' values of the face's LBP histogram.

- Block RAMs utilized to store the input data and the temporary output, bin values.
- A process resetting the BRAM temporary output values every time the IP is utilized. This is done by setting the values of the BRAM to '0' every time the core is used.
- A main process for computing the LBP values and creating the histograms is designed.
- A process of setting the temporary output values to the actual output is included.

Taking in consideration all the above, the baseline design is created and tested. A total latency of 996800 clock cycles is reported. This can be translated to an execution time of 9968000 nanoseconds(ns) for a 100Mhz frequency clock, which is the default vale for the Programmable Logic (PL) Clock. This execution time is examined on four internal sub-processes:

Process 1: Storing the input data to BRAM.

Process 2: Resetting the temporary output values.

Process 3: Computing LBP values and creating the LBP histogram.

Process 4: Setting the final output values.

As seen in Table 5.1, the main process of computing LBP values and creating the histogram is taking the most time, as expected, with 880000 clock cycles in total. The rest of the processes have the expected execution latency. Iteration latency of 1 clock cycle for both the first two processes, as one clock cycle is required for writing to the BRAM, and iteration latency of 2 clock cycles for setting the output, as two clock cycles are needed to read from the BRAM the temporary output values. The trip count, as observed, matches the number of pixels in the first and third process and the number of bins in the second and fourth process.

In Table 5.2, we can also report the resources utilization. We observe that a very low percentage of the resources is being utilized.

Table 5.2: Baseline design resources utilization

	BRAM	FF	LUT
Total	64	1319	1875
Available	530	157200	78600
Utilization	12%	~ 0%	2%

The total execution latency, as mentioned above, is equivalent to 9.9 ms. A speedup of 51.9x is reported, compared to the C code running on the ARM and the goal of 16ms is being met. However, further optimization of the IP will be examined, as the low resources utilization allows further improvement in performance. In the sections followed, a different set of optimizations will be examined, leading to the final design of the IP.

5.2.1 Modulo Operation

In Table 5.1, we report that the 3rd process is primarily increasing the total latency of the IP. This is a result of the iteration latency of 22 clock cycles. The third process has the role of Computing LBP values and creating the LBP histogram.

First, the center pixel and the neighbor pixels are fetched by the BRAM containing the input values. Next, a comparison between the center pixel and each one of the neighbor pixels is performed. This comparison leads to 8 binary values in 8 registers, as explained in 3.1.4. Those 8 registers are then combined to create an 8-bit value, LBP value. This value belongs to the bin of a region of the histogram. Therefore, the pixel region is initially located and alongside with the LBP value, the value of the correct address of the BRAM, containing the temporary output values, is fetched. Finally, this value is increased and stored back to the same address.

Through a more in-depth examination on the performance of the IP, the main source of latency is found. This is locating the region, where a pixel belongs. To locate the region, two conditional statements are used. The first conditional statement checks in which group of 10 regions is the pixel found. Those groups are formed sequentially. The first ten regions belong to the first group, the next ten to the next group, etc. This grouping is shown in Figure 5.4, where the different groups are represented with different colors.

Through the second conditional statement, the exact region where a pixel belongs, inside a group, is located. As each region inside a group covers a unique range of x-axis values, this range leads to the location of the exact region. A modulo operation is used for this purpose. The modulo operation returns the remainder of the division of the pixel index with 200 (number of pixels per row). This remainder leads to the exact region inside a group. If is in the range 0-19 is in the first region, if it is in the range 20-39 to the second region, etc. The combination of the two conditional statements locate the exact region of a pixel. The following pseudo-code presents an example of how the exact region is found:

```
i= input number;
if i<4000:
```

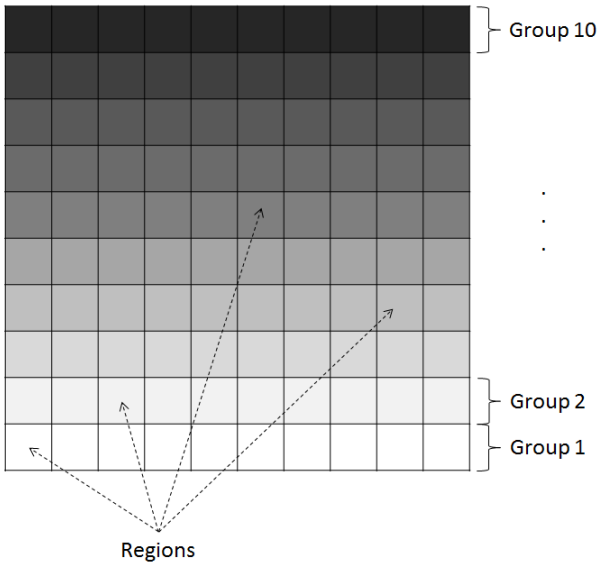

Figure 5.4: *Grouping of fist conditional statement. (Same color regions belong to the same group)*

```
    if modulo(i,200)<20:
        region=1;
    else if ...
else if ...
```

Listing 5.1: Inspecting if i-pixel is located in the 1st region

After explaining how the locator is working, the source of the latency can be spotted. The first conditional statement is a simple comparison, which is not a complex operation. On the other hand, the modulo operation on the second conditional statement, is a complex operation, on the FPGA, as it is requiring a division operand. The modulo operation utilized is the default VHDL *mod* operand. The performance analysis on the current block, showed that 19 clock cycles are needed for this operation.

To cope with this degradation of performance, resulting from the default modulo operation, a more efficient modulo operation, specific for our case, will be designed. The advantage of our model is that the upper and lower bounds of the pixel index are known, ranging from 0 to 39999. This enables many simpler solutions. Our design proposal for the modulo operation is as follows:

1. The 200-range group of the pixel index is found through a conditional statement. The number of such ranges are 200(0-199, 200-399 ... 38000-39999).

2. The pixel index is subtracted by all multiples of 200 up to 38000 and by 0.

3. A multiplexer is then utilized. The multiplexer drives the result of the subtraction that matches the right range to the output.

Table 5.3: Design performance in clock cycles with proposed modulo operation

Process	Latency	Iteration Latency	Trip Count
1	40000	1	40000
2	25600	1	25600
3	480000	12	40000
4	51200	2	25600
Total	596800		

The overview of the design described above can be seen in Figure 5.5. The range identifications and the subtractions can be all performed concurrently, making this proposal really efficient. This design decreases the clock cycles needed to compute the modulo, to two clock cycles.

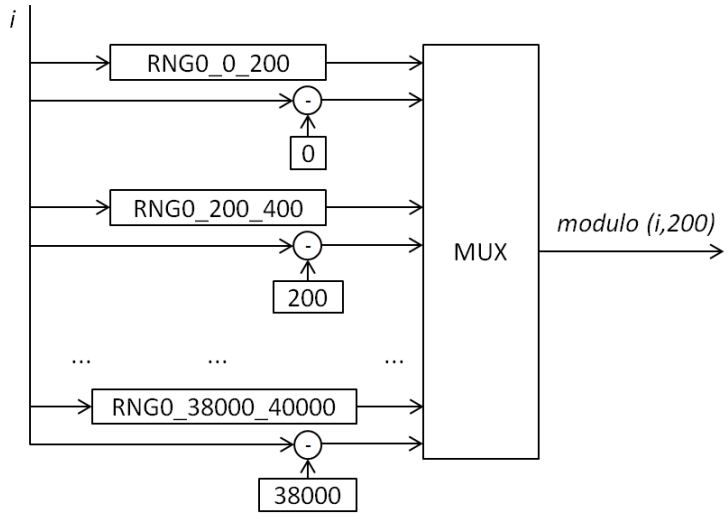

Figure 5.5: *Overview of the proposed modulo operation design.*

After applying this proposed modulo operation to the design the overall performance of the IP is increased. The total cycles needed, are now 596800, which is x1.5 times faster than the baseline design. The performance, on the sub-processes level can be viewed in Table 5.3. The resources utilization is also increased, as seen in Table 5.4, but remains still on a low level.

Table 5.4: Design utilization with proposed modulo operation

	BRAM	FF	LUT
Total	64	1614	6452
Available	530	157200	78600
Utilization	12%	1%	8%

5.2.2 Input and Output Size

An exploration on the input and output data size of the core will be done, in order to optimize the its performance further. Currently, both the input and the output consist of 32-bit words. This is a surplus form of data, as the pixels consist of up to 8 bit values and the histogram values of up to 16 bit values. Therefore 24 bits in the input and 16 bits in the output are utilized for no reason. This mainly increases the transfer time of the data from the DDR to the IP and vice versa. For this reason the input data port can be set to 8-bit and the output data port to 16-bit.

Although this is an acceptable solution, it only allows the processing of one piece of data (e.g. one pixel) per clock cycle. For this reason, we need an input and output data size that allows the processing of as many as possible separate piece of data at one time. The limit of the AXI interface [39], that we utilize, is 64 bits, which is the bus width. This size is, thus, preferred for the input and output ports to maximize the piece of data included. However, the output port that is connected to the DDR has a limitation of 32-bits, due to the DMA [40] interface which will be explained later on. The 32-bit size word is consequently utilized for the output. Therefore, 8 pixels are contained in an input word, as seen in Figure 5.6 and 2 histogram bin values in the output.

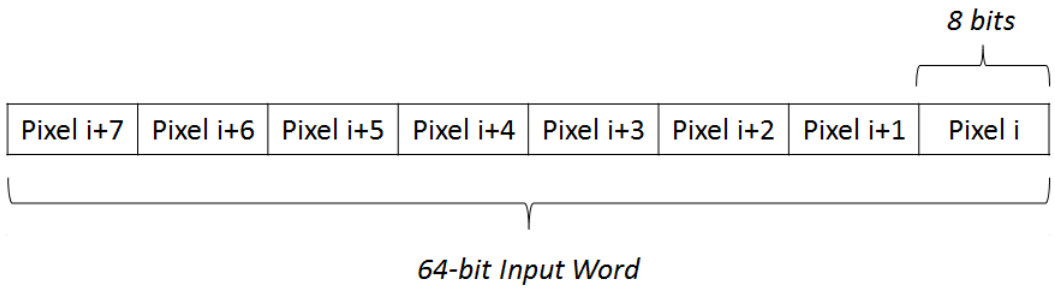

Figure 5.6: *Input Word Data Organization*

The new output and input sizes are applied in the block. The new performance is presented in Table 5.5. We notice, that the trip count in process 1 and 4, has decreased 8 and 2 times respectively. This is the result of the data contained in each word. However, the iteration latency is increased to 4 clock cycles in process 1, while it remains the same in process 4. The explanation for this fact, is found in the design of the BRAM [41], which has only 2 ports. Due to those specifications, only 2 words can be read or written to and from the BRAM. That being the case, 4 clock cycles are needed to process 8 pixels. The iteration latency of process 4 remains the same, as the bin values to be processed are just two, which can be performed concurrently by the dual-port BRAM. The utilization percentage of the new design remains almost the same, with just a slight increase in LUTs and FFs.

5.2.3 Block RAM Utilization

In the previous section, 5.2.2, the limitations of BRAM were reported. The main effects of this limitation were observed in process 1, where the iteration latency was increased.

Table 5.5: Core design performance in clock cycles with new input and output data size.

Process	Latency	Iteration Latency	Trip Count
1	20000	4	5000
2	25600	1	25600
3	480000	12	40000
4	25600	2	12800
Total	551200		

Another effect, has to do with the processes, that require multiple data, where slow fetching of the data, degrades the performance. This is the case in process 3 and will become more clear in later optimizations.

To overcome this issue, we increase the number of BRAMs utilized [1]. This way, more data can be read and written simultaneously, from and to the memory. The number of block RAMs, utilized, has to be decided. It is clear, that the higher the number of BRAMs, the more the data, that can be accessed at the same time. However, there is a limitation on the number of BRAMs in our system, 530 different 18kb BRAMs. In addition, indexing the data becomes more complex, when a high number of BRAMs are utilized.

Finding the most proper number of BRAMs, we need to take in account the way our model works. Initially, the number of BRAMs for input data will be examined. In order to decrease the iteration latency of the first process ideally, 4 different BRAMs, at least, need to be utilized. With 4 BRAMs, all 8 pixels contained in one word, can be stored at once. Second, the LBP computation is taken in consideration. 9 pixels are needed for the computation to be performed. The optimum would be for all of the 9 pixels to be stored in different BRAMs and be fetched at one clock cycle. In order for this concurrent fetching to be available in all the LBP values computation, and not just in specific cases, 600 BRAMs would be required (every pixel in three consecutive rows of the image to be stored in different BRAMs), . This can be seen in Figure 5.7, where the LBP operation can be applied at any pixel and the values needed can be fetched concurrently. However, this optimum design is not possible, as it exceeds the number of the available BRAMs (530 18kb BRAMs).

Another approach, close to the previous case, that would comply with the available resources, is to store every pixel in each row at a different BRAM. We should take in account, as the input data is imported in sets of eight, the number of the BRAMs used, needs to be a multiple of 8. However, this design would unnecessarily utilize a large number of BRAMs, as the same performance is assured through the current utilization of 4 BRAMs, and also leads to complex indexing of the data. Hence, 4 different BRAMs is a sufficient number for our design. Yet, the final number selected is 8 BRAMs, in order to simplify the development process. Instead of keeping track of 2 addresses at the same

[1] Basically, the number of the BRAMs utilized on the FPGA will remain the same, due to the data size, but what is intended is to increase the BRAMs, as seen by the developer. *BRAM partitioning* would be a more appropriate term, but we will stick to *number of BRAMs* for simplicity in explanation.

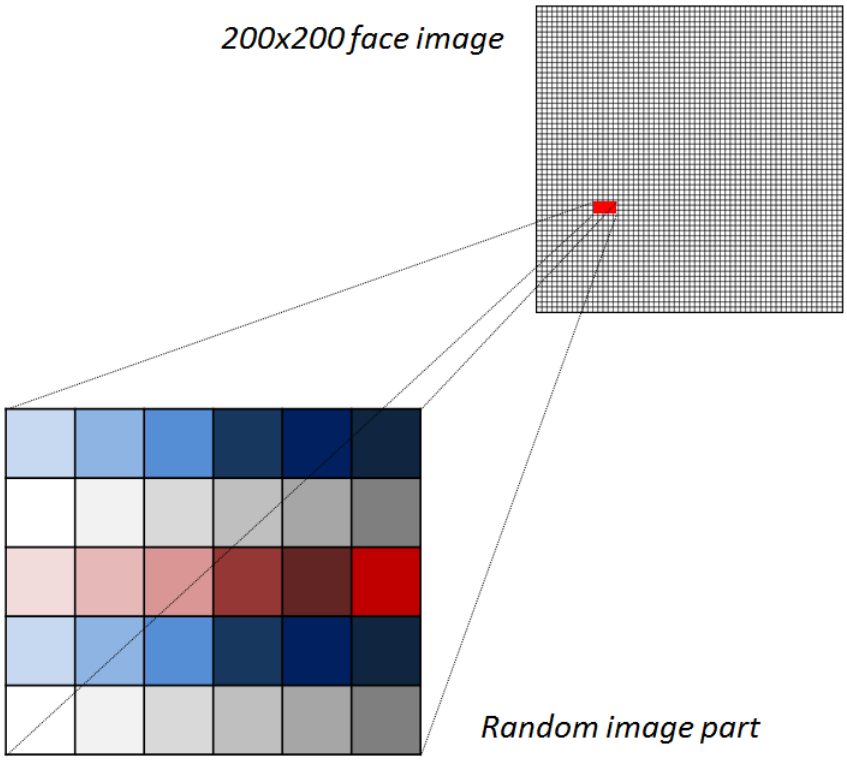

Figure 5.7: *Ideal BRAM number utilization (The different color implies the utilization of a different BRAM)*

BRAM, is preferable to keep track of 1 address at different BRAMs, from the developer's perspective. The same performance for the IP core is verified for this selection.

Concerning the number of the BRAMs utilized for the temporary output data, a utilization of more BRAMs in not required. During the process of setting the output, the maximum number of temporary output data, processed at the same time are two and this can be performed efficiently by one BRAM.

The new number of BRAMs are applied to the design and the performance is examined. Observing Table 5.6, we notice that the iteration latency of the 1st process is decreased to one clock cycle, as expected. Another remark, is that the iteration latency of the 3rd process decreased by one cycle as well. This has to do with the concurrency of data fetching, that is being enabled, as explained above.

An increase on the utilization resources is seen in Table 5.7. The bigger utilization increase is reported on the LUTs, which is now 17%. We also observe, that the BRAM number remains the same. As explained, this is expected as the number of different BRAMs does not imply different BRAMs on the FPGA, but different BRAMs as entities viewed by the developer. For example, if we would like to store 72kb on 2 *different BRAMs* (36kb on the first BRAM and 36kb on the second BRAM) the number of BRAMs used on the FPGA will be 4 and not 2 (4x18kb BRAMs). The same number would be utilized on the FPGA if we store all the data in *one BRAM*. The difference is situated in the perspective of the developer, where the concurrent fetching of data from

Table 5.6: Block design performance in clock cycles with the proposed BRAMs utilization

Process	Latency	Iteration Latency	Trip Count
1	5000	1	5000
2	25600	1	25600
3	440000	11	40000
4	25600	2	12800
Total	496200		

the *two BRAMs* is assured only in the first case.

Table 5.7: Block design utilization with the proposed BRAMs utilization

	BRAM	FF	LUT
Total	64	5014	14099
Available	530	157200	78600
Utilization	12%	3%	17%

5.2.4 Temporary Output Initialization

Process 2 is utilized to reset the temporary output values. This is done by setting the values to '0' in the BRAM, every time the IP is used. Looking at the sub processes, we note that process 2 executes, after process 1 is finished. However, the two process are completely different, as they access different BRAMs. As a result, it is wise to execute those two process concurrently to increase performance.

If the processes run concurrently, with their current performance, the execution latency of the whole IP will be reduced by 5000 clock cycles, as process 2 will start at the same time as process 1. Further improvement is possible, by optimizing process 2.

The execution latency of process 1 is optimized at the maximum level, as only one 64-bit word can be process per clock cycle. The goal is to alter process 2, in order to achieve the same or less latency. This way, process 2 will not affect the system latency at all, as it will be "hidden" under the performance of process 1.

To achieve that, the trip count has to be reduced to at least 5000, while the iteration latency remains the same. This means that 5.12 values need to be reset every clock cycle, which is translated to 6 values. To access 6 different values at the same time, 3 BRAMs are needed. While the resources allow it, 4 BRAMs are utilized for convenience, as 8 (4*2) is a divisor of 25600.

Through this modification, 8 values are reset in parallel. Every value needs 1 clock cycle to be accessed, so every clock cycle, 8 values are reset. The trip count of the process is now $25600/8 = 3200$, which is also the latency of the process in clock cycles. This value is less than the latency of process 1, 5000 clock cycles. As a result, through their concurrent execution, the total latency is equal to the latency of process 1. The new performance of the IP can be seen in Table 5.8. The total latency is reduced by

Table 5.8: Design performance in clock cycles with the proposed reset design

Process	Latency	Iteration Latency	Trip Count
1 (& 2)	5000 (3200)	1 (1)	5000 (3200)
3	440000	11	40000
4	25600	2	12800
Total	470600		

25600 clock cycles and is presently 470600 clock cycles. This improves the previous design performance by 5.5 p.p. and a total improvement by 111.8 p.p. compared to the baseline design is reported.

5.2.5 Pipelining

Another major optimization, is to pipeline our design. Pipelining a process, allows each iteration to start before the previous one is finished, while the needed resources are available. Through that, a number of tasks run concurrently reducing the execution time. A pipeline overview can be observed in Figure 5.8.

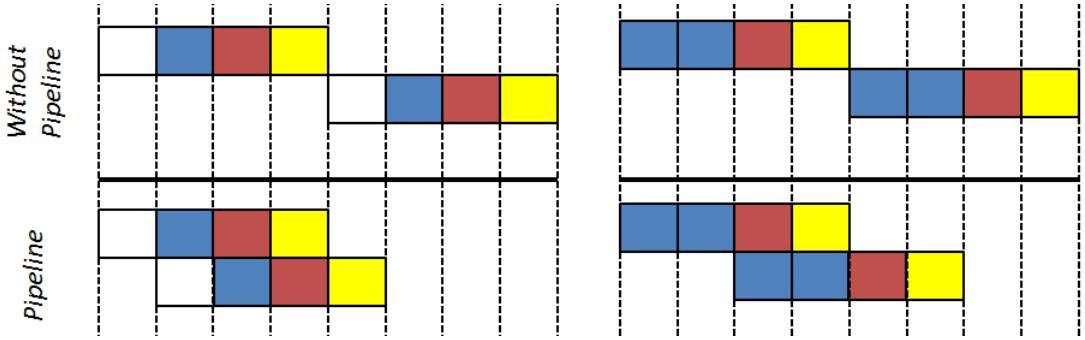

Figure 5.8: *Two (right and left) examples of pipeline application. (The squares indicate tasks and the different colors the different resources utilized)*

The two examples show different utilization of resources. In the first one, the next iteration is able to start after one clock cycle, while in the second, two clock cycles need to pass, before starting the next iteration. Furthermore, we observe the decrease of total latency by 3 clock cycles in the first example and by 2 clock cycles in the second. The total latency after pipelining can be calculated by the following type:

$$L = (N - 1) * I_P + I_L$$

, where L is the total latency, N is the number of Iterations (Trip Count), I_P is the latency intervened between two successive iterations, which will be called pipeline latency from now on, and I_L is the latency of an iteration.

The two processes, that pipelining can be applied, in the current core are process 3 and process 4. Process 3, consists of many tasks that require different resources. For

Table 5.9: Core design performance in clock cycles with pipeline applied

Process	Latency	Iteration Latency	Pipeline Latency	Trip Count
1	5000	1	-	5000
3	160004	8	4	40000
4	12801	2	1	12800
Total	177805			

example, once an LBP value is calculated, the computation of the next pixel's LBP value can start and does not have to wait for the increment of the appropriate histogram bin to take place. In process 4, the pipeline is possible as a successive read operation can start after the first cycle of its predecessor.

Pipeline is applied in both process 3 and 4. The performance is once more examined and presented in Table 5.9. The total latency is decreased to 177808, which is an 164% increase compared to the prior design. The pipeline latency of process 4 is 1 clock cycle, as expected, and for process 3 is 4 clock cycles. This pipeline latency is a result of the duration of the LBP value computation (fetching the data and computing the value) which sums up to 4 clock cycles. Another remark, is the iteration latency of process 3 that decreased to 8 clock cycles. This is the result of the decrease in data fetching, as through pipelining some data are directly available for the next iteration. The resource utilization remains at the same levels, with unremarkable differences in LUTs and FFs.

5.2.6 Loop Unroll

The seek of further parallelization, led to this final optimization. We examine how many entire processes 3, LBP and bin increment operations, can be computed completely in parallel. Concerning the LBP operation, the limitation is found on the number of pixels, required for the computations, that can be fetched simultaneously. Due to the BRAM utilization, described in Section 5.2.3, in the same time the 9 pixels for one LBP operation are fetched, 15 more pixels can be accessed. This total of 24 pixels allows 6 consecutive LBP operations to be done in parallel. This number of LBP operations available is a result of the data required overlap. An overlap of data can be seen in Figure 5.9 for two consecutive LBP operations. As seen, 12 pixels are needed for two LBP operations. Consequently, the LBP operation is available for 6 times more parallelization.

Once the LBP value is computed the right bin value of the histogram must be increased. The parallelization of this part must be examined as well. Following the factor of 6, mentioned above, we assume that the bin increment is parallelized by the same factor. The bottleneck in this part, is that the bin which must be increased cannot be predicted beforehand. Having said that, there is a possibility that all 6 bin values, that must be increased, are located in the same BRAM and thus cannot be accessed at the same time. Therefore, due to this limitation, the maximum parallelization factor allowed is 2 for the entire process.

This parallelization, is similar to unrolling the loop of process 3 by 2 times. Before proceeding with that, we need to take in account a critical case. When the bin value

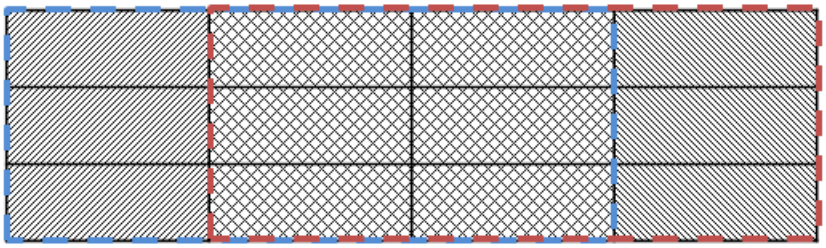

Figure 5.9: *Data overlapping of two consecutive LBP operations (blue and red border lines indicate the data for the first and second operation respectively*

to be accessed by the two parallel processes is the same. In this case, there will be a conflict, and possibly, the value will be increased just once. For this reason, we need to examine if the bin to be accessed is the same by the two processes, in order to preserve the right functionality. The design for this check operation is presented in Figure 5.10. If the bin value is the same, the value is fetched once and increased two times.

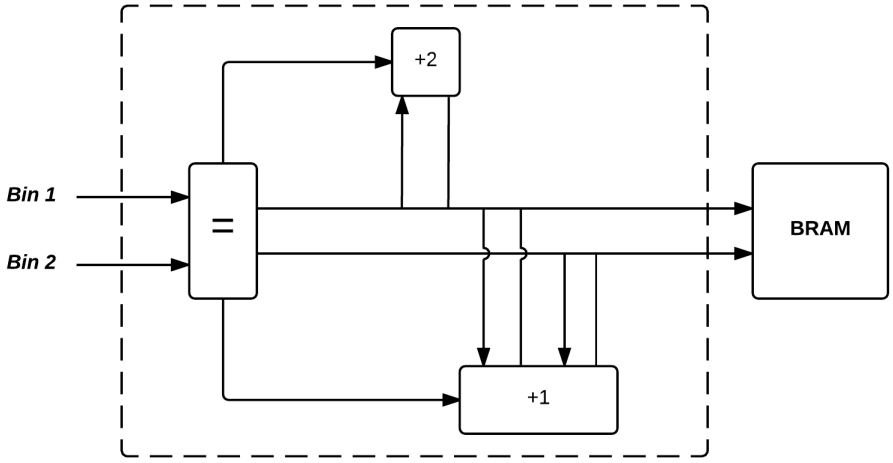

Figure 5.10: *Process to check the bin values accessed in the BRAM*

This is the final optimization for the current IP core. The performance after this final addition is presented in Table 5.10. The trip count is decreased to 20000 for process 3 as a result of this parallelization, resulting to a total latency of only 80004 clock cycles. The total utilization, as seen in Table 5.11 is increased as well, as a result of the design parallelization. Despite that, the utilization percentages remain on low levels, leaving adequate resources for the rest of the system design.

The design of the Feature Extraction IP core is complete at this point. A set of different optimizations, decreasing the execution latency, as seen in Figure 5.11, has been applied leading to a final execution latency of 97804 clock cycles. The speedup of the different optimizations compared to the baseline design is presented in Figure 5.12. This execution latency corresponds to an execution time of 978040 ns (0.978 ms) for

Table 5.10: Final IP core design performance in clock cycles (after further parallelization)

Process	Latency	Iteration Latency	Pipeline Latency	Trip Count
1	5000	1	-	5000
3	80004	8	4	20000
4	12801	2	1	12800
Total	97804			

Table 5.11: Final IP core design utilization (after further parallelization)

	BRAM	FF	LUT
Total	64	5714	22246
Available	530	157200	78600
Utilization	12%	3%	28%

the clock set to 100Mhz. This time is way under the 16 ms, which is the performance goal of our system. In addition 15.022 are still available for the execution of the rest of the application. Compared to the execution time of the feature extraction process in the ARM as reported in Section 5.1, an acceleration of 525.38 p.p. has been achieved.

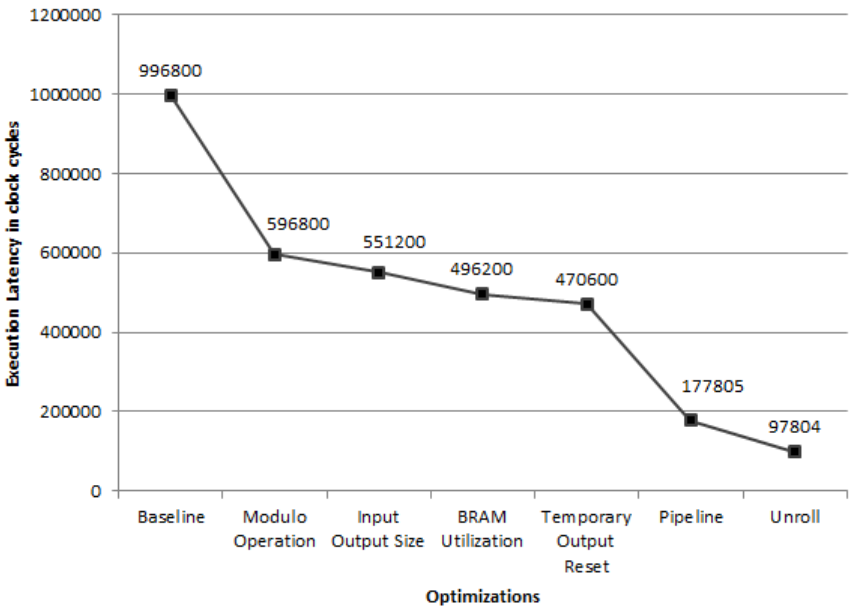

Figure 5.11: *Execution Latency of different optimizations in clock cycles*

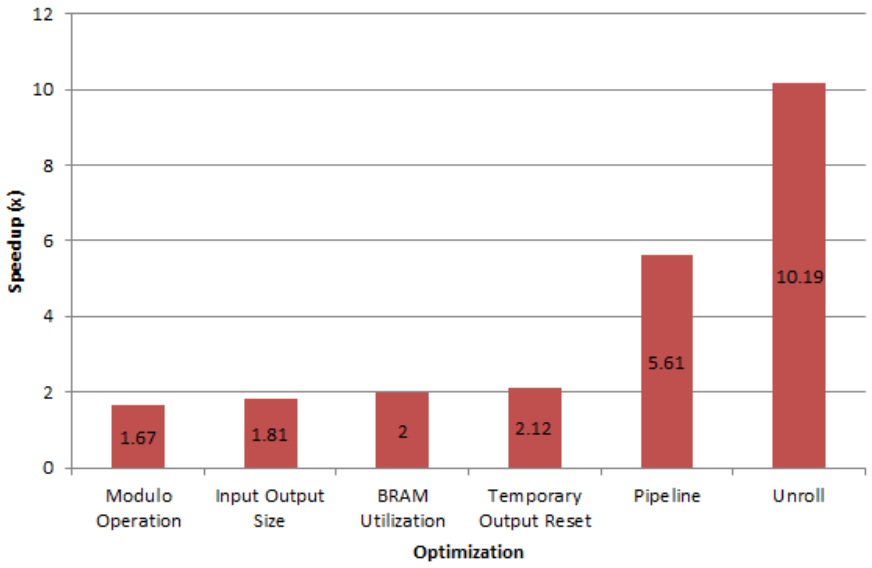

Figure 5.12: *Speedup of different optimizations compared to baseline design.*

5.3 Histograms Comparison IP core

Once the feature histogram of a test face image is generated, a comparison between the database's feature histograms needs to take place. The Feature Extraction IP core was designed effectively, Section 5.2. The Histograms Comparison IP block needs now to be designed, on the same path, to ensure high performance on the system.

As described in Section 4.3, the manhattan distance is the model this IP is based on. Through manhattan distance, a distance between two histograms is being returned. The model should be utilized to compute the distance between all 20 feature histograms of the database. The minimum distance of those 20 is returned as the best match for the test face.

To start with the design of the block, the VHDL code need to be written, based on the Matlab code developed in 4. Once more, the Vivado and Vivado HLS tools are utilized for this purpose. The main operations of the current core is first to store the inputs in the BRAM and second to compute the distance.

The optimum design for the IP will be to have the database histograms stored on the BRAM initially and from that point on, only the test faces histograms will be needed by the core, as an input to compute the distances. Although, this would save a huge amount of transactions between the RAM and the IP, it is not possible to be applied. This has to do with the limitations on the size of the BRAM, which is 9.3Mb. The size of the database together with one test face histogram sums up to 8.2Mb. As 12% of the BRAM are already utilized by the feature extraction IP, that leaves 8.18Mb available. Therefore this design approach is not possible. In addition, this approach wouldn't allow alterations in the database during run-time and limits the system's available options.

The next strategy followed is to calculate the distance between the test face histogram and one database histogram each time and store the result on the DDR. This way, 20

distances will be stored on the DDR. Finding the minimum of those distances is a very simple task and instead of designing an IP for this calculation, we could utilize the ARM. Utilizing the ARM will not degrade the performance, as it is a very simple task and also it would not complex our FPGA design further.

Another aspect that should be examined is the form of the input. One method will be to merge the database histogram and the test histogram and import both of them as one input. This would require only one transaction between the DDR and the FPGA, benefiting the performance. However a copy of the test histogram in the DDR will be needed before every transaction. An investigation on this copy operation, showed that the time needed is about 6ms, which is a major addition on the execution time. For this reason, two inputs are used in the IP, as the two transactions would be more efficient. The two approaches described can be better comprehended by observing Figure 5.13.

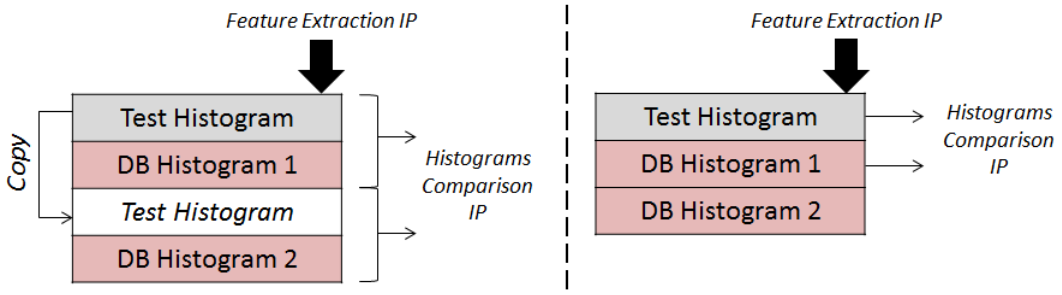

Figure 5.13: *Import input methods. (Left: One unified input, Right: Two inputs*

The baseline design can now be implemented. The AXI interface is utilized again for both the two input and the output. The inputs consist of 16-bit words and the output of a 32-bit output. The baseline design is consisting of two main processes. The first one is used to save the input data to the BRAM and the second one is using those data to compute the distance. We shouldn't forget the process of resetting the temporary output result in every utilization, as the values are added on top of that. The same process as in Feature Extraction block is followed, by setting the value to '0' at every IP utilization. However, in this IP, this is a minor process as it regards only one value.

Concerning the distance calculation between two histograms, a number of iterations is needed to cover all the bin values of the histograms. At every iteration, the two different histogram values are loaded from the BRAM. Then, the Manhattan distance is calculated and the proper region-weight is added. The new value is added to the total distance. Finally, when the iteration is finished the total output is returned. This whole process is presented in Figure 5.14.

The overall performance results of the baseline design can be viewed in Table 5.12. We observe that the trip count of the first process is 25600, as two values are included at every input word. The iteration latency of the second process is 4 clock cycles, composed by fetching the two values needed, compute the distance with the applied weight and add it to the total distance. The overall performance is 128000 clock cycles, translated to 153600 ns for the default frequency of the clock. As this process needs to be repeated 20 times for the whole database, the overall execution time adds up to 30.72 ms. Although this is a significant improvement, compared to the execution time, 899.598 ms, of the C

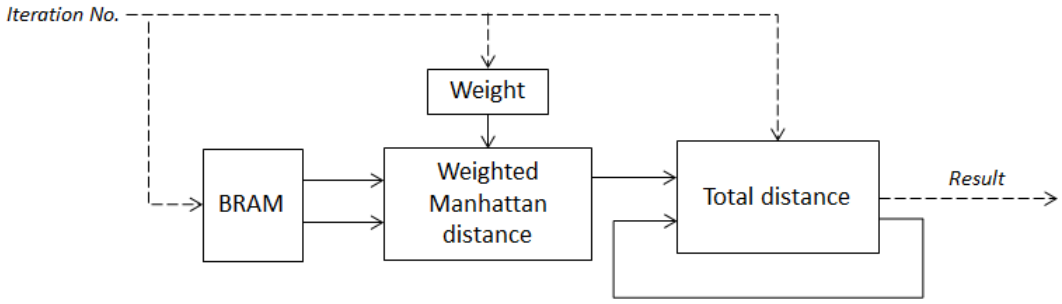

Figure 5.14: *Overview of two histograms' distance calculation*

Table 5.12: Histograms Comparison core baseline design performance in clock cycles

Process	Latency	Iteration Latency	Trip Count
Store Input Data	51200	1	51200
Reset Temporary Output	1	1	1
Distance Calculation	102400	4	25600
Total	153601		

code in the ARM, it is not acceptable for our system as it exceeds the real-time goal of 16ms. Further optimization approaches are needed and will be explained in subsections 5.3.1-5.3.5. The low percentage of the resources utilization, presented in Table 5.13, gives a large margin for the optimized design.

5.3.1 Input Data

The main optimization of our design is based on the idea described before, to include the whole database in the BRAM. This would decrease the transactions of data between the DDR and the IP and in addition, as more data will be available, parallelization possibilities will be increased. Those benefits are desired for our design, but as this implementation is not feasible, alterations should be made.

As the whole database size is exceeding the BRAM size, we proceed by dividing it in two parts, composed by 10 histograms each. The size of each part is 3.9Mb, which is less than half of the total BRAM size. As the size of every part allows its storage on the BRAM, the core is designed to process the test histogram and one half of the database. Therefore the inputs are now the 25600 values of the test histogram and the

Table 5.13: Histograms Comparison core baseline design resource utilization

	BRAM	FF	LUT
Total	64	133	1005
Available	530	157200	78600
Utilization	12%	$\sim$ 0%	1%

Table 5.14: Design performance in clock cycles by importing the half database.

Process	Latency	Iteration Latency	Trip Count
Store Input Data	281600	1	281600
Reset Temporary Output	10	1	10
Distance Calculation	1024000	4	256000
Minimum Distance Calculation	20	2	10
Total	1305630		

256000 values of the 10 histograms of the one half of the database.

The first benefit from this implementation, is the transactions time needed. The 25600 values of the test histogram require now to be transferred only two times to the IP. Those are 18 transactions less compared to the baseline design, improving this way the performance of the system.

The distance between all 10 histograms is now calculated. 10 temporary output values are now utilized that need to be reset as well at every IP utilization. Moreover, as those distances are available on the core, we can also include in the design a process to calculate the minimum one and return only one distance to the DDR. This process is simple to design on the FPGA and will reduce the calculations needed by the ARM in the end. The ARM's operation is now limited to computing the minimum between two distances, one from each half of the database, increasing further the performance. Alongside with the minimum distance value, the index of the histogram corresponding to that distance should be output, otherwise recognition would not be possible. Hence the output of the core is now 2 32-bit words, one for the distance and one for the index.

The performance of the modified design is presented in Table 5.14. The increase on the iterations number of the process is obvious and is a result of the increase of the data. The total execution latency of the design is 1305630 clock cycles. This execution latency multiplied by two, shows the total execution latency of the IP, as it is utilized two times to cover the whole database. This latency is equal to 2611260 clock cycles. This decrease is a result of the less BRAM store operations, needed for the test histogram. The increase of the total system performance, due to the less number of transactions, cannot be seen through those results, but will be seen when the IP is integrated in the final system. The latency, which is corresponding to an execution time of 26.11 ms, is still not acceptable for our system. Concerning the utilization, a large increase, reaching 58%, on the percentage of BRAM utilization is reported, as expected. The FF and LUT utilization has slightly increased due to the introduction of new processes.

5.3.2 Input Size

The process of maximizing as much as possible the input size, as followed for the feature extraction IP in Section 5.2.2, will be followed in the design of this core too. The output data size is not concerning us in the current IP, as it consists of only 2 words. This minor value cannot effect the performance of the core and thus the default output size of 32-bit words is used.

Table 5.15: IP design performance in clock cycles with new input size.

Process	Latency	Iteration Latency	Trip Count
Store Input Data	140800	2	70400
Reset Temporary Output	10	1	10
Distance Calculation	1024000	4	256000
Minimum Distance Calculation	20	2	10
Total	1164830		

In more detail about the input size, the maximum size that can be processed at once is 64-bit words. The size of the bin values is 16 bit, which means that 4 values can be included in on 64-bit words. This would be the size that is going to be used for both two inputs. The format of the input is presented in Figure 5.15.

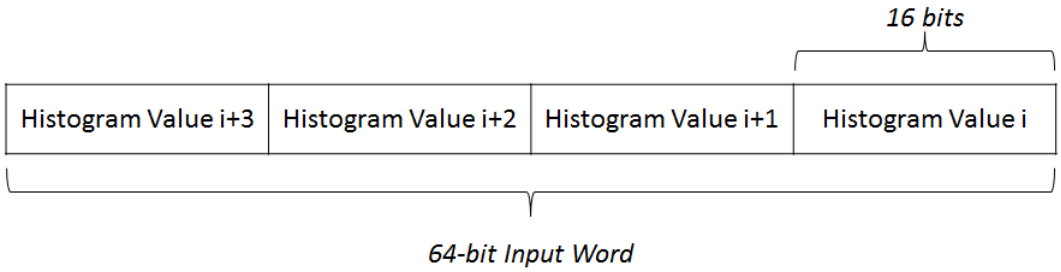

Figure 5.15: *Input Format*

The benefit of this optimization is seen in two parts of the system. The first one is in the current IP and the fact that more data are available for processing per clock cycle. The second part has to do with the transfer of data from the DDR, as through this size, the transaction latency is decreased, due to the burst type transactions that will be explained later. The new performance is presented in Table 5.15. The trip count for the first process has been reduced by 4 times, as expected. However, the iteration has been increased to 2 clock cycles as only two values can be stored at the BRAM at one time.

5.3.3 Block RAM Utilization

In the previous subsection, 5.3.2, the limitation of using one BRAM was observed, as only two values could be stored at once, although 4 values were available. This issue can be resolved by utilizing more BRAMs as performed in the Feature Extraction core described in, Section 5.2.3.

The number of BRAMs utilized is decided based on the solution of the mentioned issue, of storing all the available data at one clock cycle. 2 BRAMs are adequate of storing all the 4 values of the input words at one time. However, a utilization of 22 different BRAMs is proposed. The reason for this utilization number is to store the different histograms' values in separate BRAMs. As the different histograms are 11 in total and

Table 5.16: Core design performance in clock cycles with proposed BRAM utilization.

Process	Latency	Iteration Latency	Trip Count
Store Input Data	64000	1	64000
Reset Temporary Output	10	1	10
Distance Calculation	102400	4	25600
Minimum Distance Calculation	20	2	10
Total	166430		

Table 5.17: Core design utilization with proposed BRAM utilization.

	BRAM	FF	LUT
Total	320	1951	7781
Available	530	157200	78600
Utilization	60%	1%	9%

2 BRAMS are required for each one to store the available input data concurrently, 22 BRAMs are needed.

Through this utilization, the parallelization of the distance calculation process is being enabled. Instead of computing the distance between every histogram, one after the other, all of the distances can be computed simultaneously as the different histogram values can be accessed concurrently.

Furthermore, this utilization enables the storing of the test histogram at the same time as the database histograms. Because of the two different input ports the data of the test histogram are available separably. This separation of input data can now be exploited as the values are stored in different BRAMs. The main result of this optimization is that the execution latency of storing the test histogram values, is kept out of sight by the latency of storing the database histograms, as those two processes happen concurrently.

We should mention, that the utilization of BRAM should not be considered for the temporary output values for this IP. The temporary output utilized is consisting of 10 values. FPGA registers are utilized for storing those values instead of BRAM. This way they can be accessed simpler and faster by the different processes.

The performance of the design after the addition of this modification is presented in Table 5.16. We note that the iteration number of the first process has been reduced to 64000 and iteration latency to 1 clock cycle, as desired initially. The major, though, improvement noticed is the trip count decrease of the third process to 25600, maintaining the same iteration latency. This leads to a total latency of 166430 clock cycles, consequently to 332860 clock cycles for the whole utilization of the IP. The execution time for this latency is 3.33ms, which is finally under the real-time performance limits. The system performance requirements have been met at this point, however further optimization would be examined as the low resource utilization, reported in Table 5.17 allows it.

5.3.4 Pipeline

The next optimization is to pipeline our design to raise further the performance. The benefits of pipelining were described in Section 5.2.5, and a similar procedure is followed in the current core. Moreover, as the operations required are simpler and more independent, in terms of data required, we expect a higher increase in performance than in the Feature Extraction IP.

In more detail, looking at the processes of the current design, the distance calculation process is the one that can profit from pipelining. The store input data and reset temporary output processes have already an iteration latency of 1 clock cycle and thus, there is no point to pipeline them. The minimum distance calculation process, although it can benefit from pipelining, it will lead to a maximum improvement by 10 clock cycles. As this improvement is minor, we won't pipeline the current process.

The distance calculation process, has a iteration latency of 4 clock cycles and a considerable iteration number of 25600 iterations. Applying pipeline, could lead to a notable improvement on the performance. Looking back to Section 5.3 and Figure 5.14, we observe that in every iteration the two values to be compared are loaded, then the distance is calculated and finally is added to the total distance. As the data to be loaded at every iteration are completely different and as the computations are depending only on these data, a pipeline latency of 1 clock cycle can be achieved. This way, every iteration can start one clock cycle after its previous iteration. The pipeline process can be seen in Figure 5.16, and we can comprehend how the execution latency is being reduced.

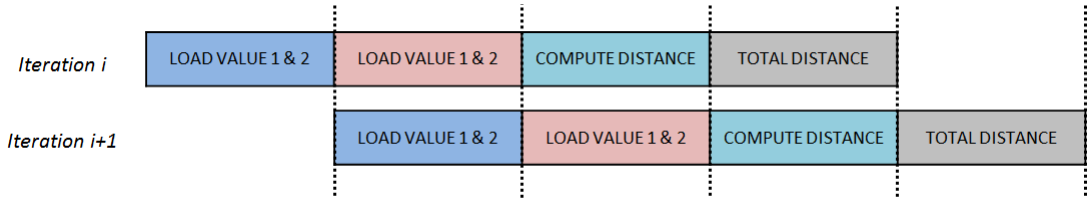

Figure 5.16: *Distance Calculation Pipeline*

The new performance after application of pipeline are presented in Table 5.18. The process pipelined has now an execution latency of 25603 clock cycles, about 4 times less than previously. The total execution latency is also reduced to 89633 clock cycles. This is a x2 acceleration compared to the prior design. The utilization is not effected noticeably by the pipeline design, as the same resources are utilized, and only slight differences in number of LUTs and FFs are reported.

5.3.5 Loop Unroll

The final optimization is centered on parallelization. Through this optimization we will examine the parallelization limits of the current IP, in order to reach the optimum performance. This is done by executing more iterations per process at the same time and thus is can also be defined as loop unroll optimization.

The process of store input data cannot be parallelized, as one word is available per clock cycle and thus the next iteration cannot execute at the same time. The same

Table 5.18: Design performance in clock cycles with pipeline applied

Process	Latency	Iteration Latency	Pipeline Latency	Trip Count
Store Input Data	64000	1	-	64000
Reset Temporary Output	10	1	-	10
Distance Calculation	25603	4	1	25600
Minimum Distance Calculation	20	2	-	10
Total	89633			

applies for the Minimum Distance Calculation as every iteration requires the minimum value calculated by the previous iteration. In order to parallelize this process, a new design should be implemented, but as the total execution time, only 20 clock cycles, is insignificant for the total execution time of the IP we will not deal with this process more.

The loop unroll optimization will be first applied to the reset temporary output process. This is done by just setting all the registers to '0' concurrently. The improvement might be negligible, but as the implementation was simple and it does not complex the design we went through with applying it.

Parallelization can be applied and improve significantly the distance calculation process. In this process as mentioned before, Section 5.3.4, the data utilized in each iteration are different and further more the computations do not depend on the other iterations. Consequently, we will seek parallelization of as many iterations as possible. The main limitation of the parallelization range is coming from the data that can be fetched for the required computations simultaneously. The number of different BRAMs utilized per histogram are 2. Hence 4 different values can be exported at the same time, leading to the factor of parallelization of 4. The distance calculation iterations are this way reduced to 6400. Also, we should mention that the iteration latency is now increased by 1 clock cycle, as the 4 different distances calculated, need to be also added together to form the total distance. However, this increase is not effecting the total execution, as the pipeline latency is remaining to 1 clock cycle. By applying parallelization on the process the execution latency of it is reduced to 6404 clock cycles, a four time improvement as expected.

As seen in the previous paragraph, the main limitation of this optimization is a result of the limited concurrent data access. To overcome this issue, the BRAM utilization, described in Section 5.3.3 will be altered, so more data can be accessed at once. Initially, we double the different BRAMs and accordingly the factor of parallelization. The process latency is once more reduced to half, reaching 3204 clock cycles. The utilization of FFs and LUTs is increased as the parallelization increases, and is now 5% and 24% respectively.

As the resources allow it, we proceed to a further increase of the different BRAMs, to 8 different ones. The factor of parallelization is now 16 leading to a process execution latency of 1604 clock cycles. The resources utilization is increased again, with 9% of FFs and 46% of LUTs. Although this is a big increase of utilization, there is margin for

further optimization. For this reason, 16 different BRAMs are utilized next, increasing the factor of parallelization to 32. The iteration latency is now 804 clock cycles. However, by examining the resources utilization, the percentage of LUTs is presently 89%. This is prohibitive for our system, as the LUTs utilization of the previous IP is 28%, leaving a 72% available for the rest of the system. For this reason, we need to move back to the parallelization factor of 16, as it is the one below the 72% boundary. The LUTs utilization increase for the different factors are presented in Figure 5.17.

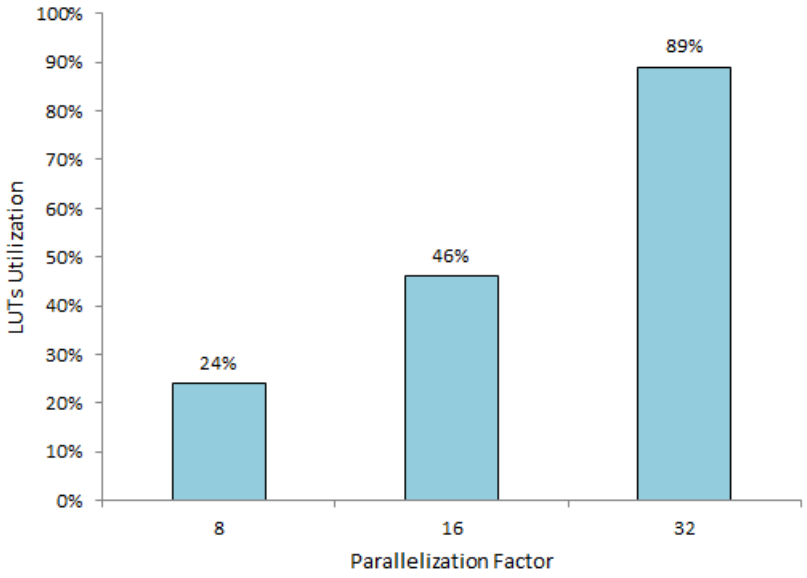

Figure 5.17: *LUTs utilization for different parallelization factors.*

The performance of the IP, after the applied parallelization, which is also the final design of the block, is presented in Table 5.19. The final execution latency of the IP is 65625 clock cycles and as the IP is utilized twice, to perform the total database histograms comparison, the overall utilization latency is 131250 clock cycles. The execution time of the Histograms Comparison block for a clock frequency of 100 Mhz is therefore 1.3 ms. This latency is way below the goal of 16 ms, ensuring this way real-time execution. Moreover, a 690.99% acceleration is reported, compared to the initial performance of the histograms comparison function in the ARM.

The utilization of the IP is reported in Table 5.20. The resources were utilized to the extent it was allowed to optimize the design as much as possible. The resources utilization should also comply with the existing utilization of the Feature Extraction IP, so that the combined utilization would not exceed the system available resources.

The process of designing the Histograms Comparison core leading to the final design has been described extensively. The purpose of this design was to achieve real time performance of the comparison process. A number of different optimizations were utilized for this reason. The further optimization of the IP, beyond real-time execution, was also searched for as the resources available allowed it. In Figure 5.18, we can observe the decrease of the execution latency through the application of a different set of

Table 5.19: Final IP core design performance in clock cycles (loop unroll applied)

Process	Latency	Iteration Latency	Pipeline Latency	Trip Count
Store Input Data	64000	1	-	64000
Reset Temporary Output	1	1	-	1
Distance Calculation	1604	5	1	1600
Minimum Distance Calculation	20	2	-	10
Total	65625			

Table 5.20: Final IP core design utilization with proposed BRAM utilization.

	BRAM	FF	LUT
Total	320	15583	36534
Available	530	157200	78600
Utilization	60%	9%	46%

optimizations. The speedup achieved, compared to the baseline design, is presented in Figure 5.19, showing the effectiveness of the optimizations. The final design of the core is completed and ready to be integrated in the system.

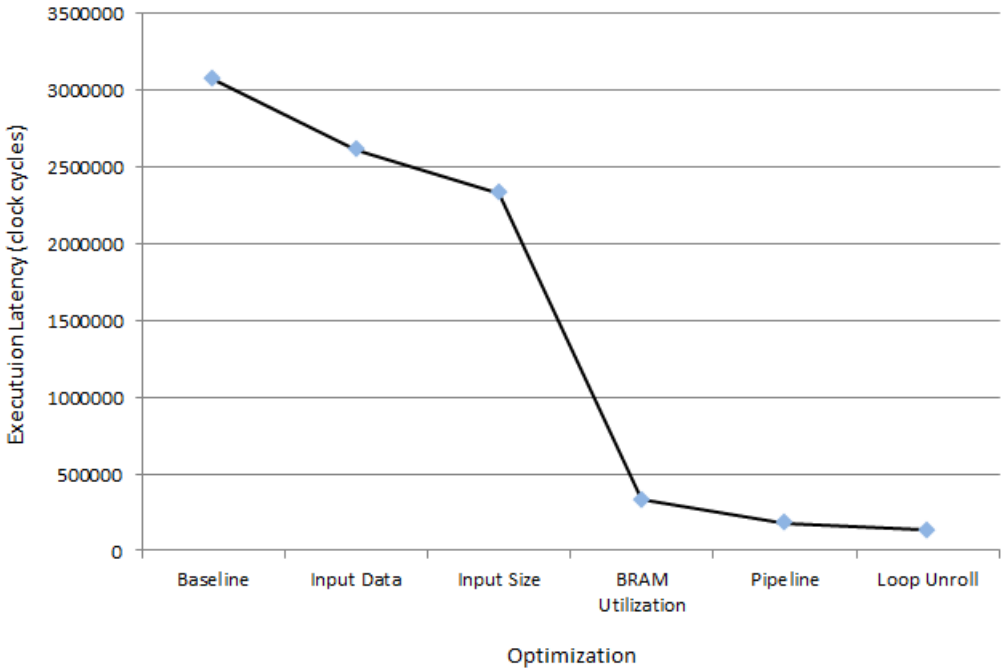

Figure 5.18: *Execution Latency in clock cycles for different optimizations.*

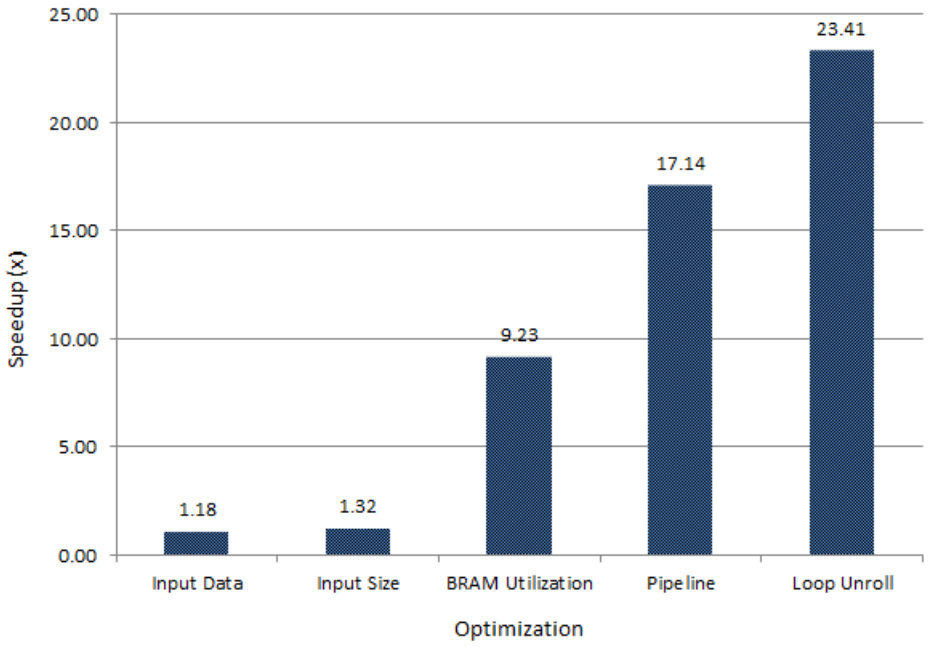

Figure 5.19: *Speedup of different optimizations compared to baseline design.*

5.4 Programmable Logic integration

The two cores essential for our system have been designed and described in Sections 5.2 and 5.3. Our system has now the ability to extract the feature histogram of a test face image and compare it to the database histograms. However, a complete integration of the cores in the FPGA is needed, in order to be functional in our system. The integration of the IP cores into a functional FPGA design has been completed using Vivado.

The first step needed is to manage the signals mandatory for the IPs' operation, clock and reset. The clock signal of the two blocks is connected to the Programmable Logic Fabric Clock of the Zynq. This port is found in the Zynq IP which is utilized in our design. This IP is essential as it handles all the main signal of the Zynq and ensures correct communication with the ARM. In addition, the reset signal is connected to the general purpose reset signal of the Zynq. This way, the functional signals for the two IPs are set.

Second, the connection of the cores with the memory needs to be established, in order to access the data needed. As described above, the AXI interface is utilized for these transactions. The AXI Direct Memory Access (DMA) [40] is utilized by our system to handle the transactions and maintain high performance. The operation of the DMA is depended on the ARM. The ARM instantiates a new DMA transaction (read or write) by indicating a start address on the DDR and the size of bytes to be written. For the DMA to work properly, though, an extra signal, TLAST, is needed for every word transferred. This signal is needed in order to determine the last word of the transaction. If this signal is missing the DMA will raise an error signal and halt his operation from that point on. Therefore this signal is mandatory and should be included in our IP design.

This adjustment is simple in our design by just adding one more port for the TLAST signal. During the set of the output data, in the last word the TLAST signal is raised and the DMA transaction can be successfully performed.

The DMA is connected through the High Performance bus with the DDR and the IP cores. The ports used for that are the S2MM (Stream to Memory Mapped), to transfer the output data of the IPs to the DDR, and the MM2S (Memory Mapped to Stream), to import the data from the DDR to the IPs. Also, it is connected with the Zynq Master GPO port. This connection is needed to indicate to the DMA about the transactions to be performed. Furthermore, the DMA has a set of interrupt signals. Although, it is not mandatory, those signals are connected to the Zynq Fabric Interrupt Ports. This is done to have a complete overview of the DMA operations and be able to control the functionality of our system better.

Three DMAs are utilized in our system. One responsible for the input and output data of the Feature Extraction core, one for transferring the test feature histogram to the comparison block and the last one for importing the DB feature histograms to the comparison block, too, and sending the results to the DDR.

The other IPs utilized are the concat IP, which is used to merge the interrupt signals of the DMAs into one, as the Zynq has only one interrupt port, a processor system reset IP, that applies properly the reset signal retrieved from the Zynq to the other IPs and finally two AXI Interconnect IPs. Those last two IPs are used as an interconnect to the DDR and to the Zynq. The first one is used for the connection between the DMA and the GP) port of the Zynq and the second one for the connection between the DMA and the DDR. Our complete FPGA design is now finished. This design is presented in Figure 5.20.

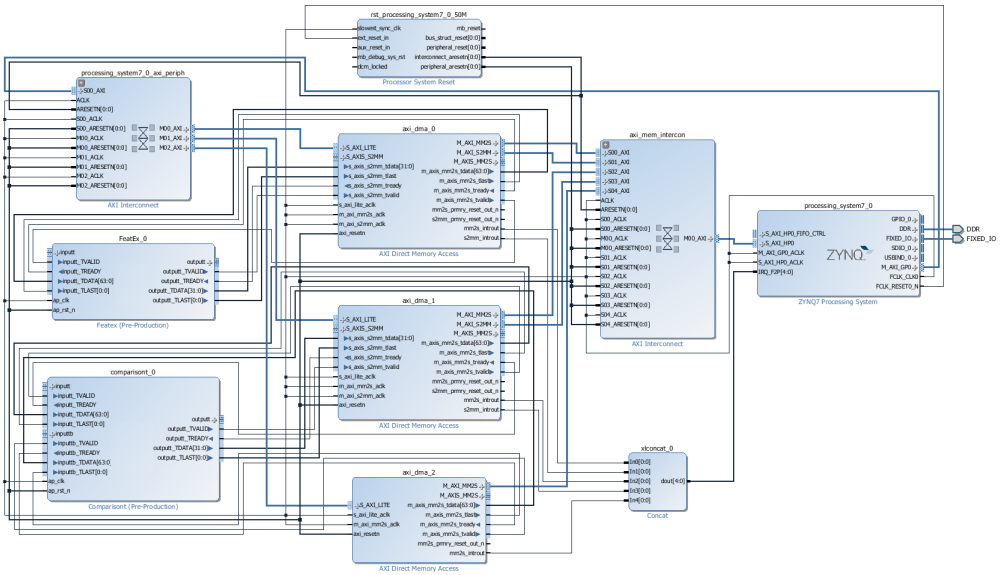

Figure 5.20: *FPGA design of our system*

The FPGA design that can perform the operations needed for our system in now complete. However, available optimizations on the design level will be investigated. The

first one has to do with the operation of the DMA IP. The DMA performs burst-based transactions. For every burst, some signals of the DMA are also modified, requiring 6 and 39 clock cycles for the MM2S channel and the S2MM respectively [40]. To improve the performance, the burst size must be set to maximum, to reduce this latency operations added to every burst. The maximum burst size allowed is 256 words per burst and should be the optimal value for the DMA operation. However, the different DMA burst sizes were tested to examine the performance. The results can be seen in Figure 5.21. Through this test, we observe how the performance improves for the higher burst size. The performance, though, reaches its peak for the 32 burst size and remains the same for the higher burst sizes, while we expected a continuous increase. The explanation for that is found on the specification of the AXI interconnects [42]. The maximum burst size allowed for transactions with the DDR is 32 words and any higher value utilized will be modified to 32 words size burst. Therefore, the burst size utilized is 32, as it ensures the highest performance, almost 6 times higher than the size 2 burst.

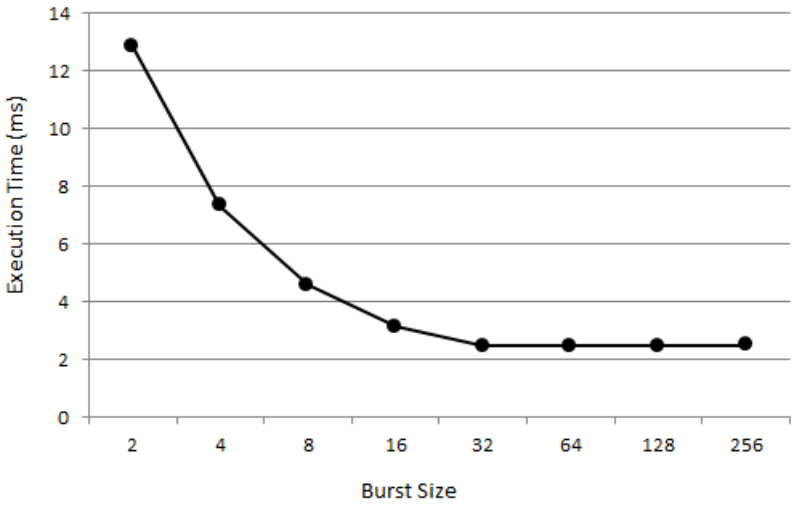

Figure 5.21: *Performance of different burst sizes.*

The next optimization, is regarding the PL fabric clock. All the operations of the IP and the DMAs are based on the clock signal. A clock signal of a smaller clock cycle (higher frequency), would lead to more operations in less time, increasing therefore the performance. However, there is a limitation on the increase of the frequency allowed. The critical path of the design should be taken in account to estimate the minimum clock cycle duration allowed. Through the Vivado analysis tools, we note that the estimated allowed minimum duration of the clock cycle is 8.95 ns, which is corresponding to a frequency of 111Mhz. The available values for the clock frequency are discrete, and the option that approaches more 111 Mhz, without exceeding it, is 109.09 Mhz. The implementation of the design is implemented successfully for this frequency. As the allowed clock cycle duration is based on an estimation, we will increase the clock frequency to the next value, 114Mhz, and examine if the time constrains are met. The implementation is once more successful and thus the next value, 120 Mhz, is examined well. The increase to 120

Mhz is unsuccessful and therefore, we step back to the frequency of 114 Mhz. This is the frequency selected for our clock, and as the the performance is depended linearly on the increase of the frequency, an optimization of 14% is achieved for our system. We should point out, that during synthesis and implementation on the Vivado, the post-place and post-route physical optimization design options were enabled to make our design more efficient.

The complete design on the FPGA, with the described parameters to achieve maximum performance, has been completed. Through this design, the face image stored in the DDR is retrieved and after extracting its feature histogram and comparing it to the database histograms, the two best matches for the two halves of the database are returned in the DDR.

5.5 Operation of the ARM processor

The needed utility of the ARM has been mentioned above, many times. The main operation of the ARM, regarding the system functionality, is computing the best match from the two returned matches by the PL. This is a very simple function to develop on the ARM and be executed effectively. The values in the DDR returned by the PL are two distances and their corresponding indexes. A conditional statement checks the smaller of the two distances and depending on the outcome the matching index is returned as the best match.

Next, as described, the DMA transactions are initialized by the ARM. 8 different transfers are performed, 5 from the DDR to the IP cores and 3 from the IPs to the DDR. Those transactions are the following: transferring the face image data to the feature extraction core, returning the feature histogram to the DDR, 2 transfers of the feature histogram to the Comparison IP, 2 transfers of the half database histograms to the same core and finally 2 transfers of the results of the Comparison core to the DDR. Furthermore, the ARM manages the addresses accessed of the DDR to avoid access of wrong addresses by the different transfers.

Except from those two essential operations, the ARM is used for a set of operations that are required for the correct functionality of our system. The first one is to initialize the whole platform at the start of the operation. A function is provided by Xilinx and utilized for this reason. Related to that is the operation of cleaning up the platform and the end of operation. A default function is utilized here too. Finally, before initializing transactions, the DMA IPs need to be enabled. This is another operation performed by the ARM, which basically writes to the control registers of the DMAs to enable them and also enable interrupts.

5.6 Conclusion

In this chapter, the complete process of implementing our system has been described extensively. In Section 5.1, our design approach was described, including the profiling of the models. After that, the two IP cores needed for our system, Feature Extraction IP and Histograms Comparison IP, were designed and optimized efficiently. The process of

doing that for the two cores is described in Sections 5.2 and 5.3 respectively. Once the IPs were completely designed, their integration on the PL of the system took place and was described in Section 5.4. Finally, the role of the ARM processor on the operation of the system, although limited, was presented in Section 5.5.

The final system designed, through this implementation, meets all the requirements that were expected. High accuracy, real-time face recognition is expected from the operation of the current system. The added latency of the two IP cores, that are the main part of the system, indicate an execution time of approximately 2 ms for our system. This way, a single face image can be recognized, through the database, in much less time than the 16 ms, which was our initial real-time goal. The accuracy results presented in the previous chapter, derived from the models utilized, are expected for our system too, ensuring high accuracy.

The combination of high accuracy and real time execution, indicates the efficiency and usefulness of this novel implementation. Through this proposed system, face recognition utilizing LBPH, which is popular for its efficiency, has benefited from the FPGA technology to achieve real-time performance.

6 Results

In this chapter, a set of experiments will be carried out, to derive results concerning the operation of our system. The implemented system, described Chapter 5, needs to be examined, in order to verify its correct functionality and evaluate its performance. In Section 6.1 the experiments performed will be described. Then the detailed results will be presented. Those results will concern execution time, Section 6.2, accuracy, Section 6.3, resource utilization, Section 6.4, and finally power consumption, Section 6.5.

6.1 Description of Experiments

The system needs a database in order to operate and also a set of test images, to evaluate its operation. The Unconstrained Facial Images (UFI) Database [33] was utilized to acquire a set of face images for our database and for testing. The UFI database is a real-world database that contains images extracted from real photographs acquired by reporters of the Czech News Agency. Therefore, the database contains images of people with different expressions, at different time and with different environmental conditions. This makes the recognition process more difficult and hence is a good evaluation choice for our system.

Five images for 20 different people will be chosen from UFI to create our database. For every person, the LBP Histograms will be extracted and added together, as needed for our system. The added histograms for every person and will be stored in the DDR of the system. An example of the images utilized for 4 people of the database can be seen in Figure 6.1.

Once the database is imported, the test images should be used. The test images should be from the same people in the database. Example test images, from the 4 example faces seen in Figure 6.1, are presented in Figure 6.2. However, we would like to have 100 test images, an average of 20 per person, to have a wide range of results to evaluate the system. UFI, though, does not contain that many photos per person. Therefore some test images were created by editing existing images. This editing contained rotation, contrast modification, edge sharpening, etc. An example of edited images through one UFI image is seen in Figure 6.3. This way a set of test 100 images are utilized to test our system.

Furthermore, to extend the evaluation, we repeat the same process two more times, import a new database of 5 other people and utilize 100 other test images. In the end, we have a set of experiments that test 300 images resulting in a better evaluation of the system.

The input and output methods of the test experiments should be defined. The input test images will be stored to the DDR through the SD card of the platform [14]. In our test application, a loop will be used to load each image from the SD card to the DDR,

Figure 6.1: *Face images of 4 people used in the database [33]*

Figure 6.2: *Example Test Images [33]*

forming this way our input stream. The issue here is that the images should be stored in the SD card in the right format. As this is a bare metal application, there is not a predefined function to read the pixel values from the image PNG format. Therefore, the images should be converted to binary format, containing the pixel values, before being stored in the SD card. A matlab function is developed for this reason and the test images are converted to binary files. Concerning the output, a message will be printed in the terminal console, indicating the best match for the current picture. The match will be shown in a number, 1-20, matching one of the 20 people. An example of output in the console can be seen in Figure 6.4.

The execution time, alongside with the accuracy, are the most important aspects of our system. Although an insight of the performance has been given during the implementation process, the actual system execution should be evaluated. This is done by utilizing the interrupt signals from the DMAs and timing points in the code of our application, to compute specific operations duration. The overall performance can be examined correctly with those measures used.

The resource utilization of the system, together with the power consumption are not

Figure 6.3: *Editing of an UFI image [33]*

```
a1 matches person: 1
a2 matches person: 1
a3 matches person: 1
b1 matches person: 2
b2 matches person: 8
b3 matches person: 2
```

Figure 6.4: *Result Output*

a primary concern for the defined project specifications. In spite of that, a brief summary on those aspects will be presented as well, to have a spherical view of the implemented system. The tools of Vivado will be used for this reason, as they can give a very good estimation for these values.

6.2 Execution Time

The performance of the implemented system will be examined first. The DMA interrupt signals are used to find the latency of the two programmable logic processes, feature extraction and histograms comparison. Timing points in our code will indicate the duration of the matching process on the ARM and the total execution of the application.

One test image is used for measuring the execution latency. After its importation in the DDR, the execution time for recognition is presented in Table 6.1. This latency matches the expected results, discussed in the implementation process (Chapter 5). We can now confirm, the real-time facial recognition of our system. All of the processes report small execution times, 0.9ms for feature extraction, 1.49 for histograms comparison, 0.04 for computing final result and add up to a real time latency of 2.43 ms.

The execution time of our system for a stream of multiple input images is linearly depended to the single-image latency. Therefore, the latency (L) for a stream of N input

Table 6.1: System execution time

Process	Latency (ms)
Feature Extraction	0.9
Histograms Comparison	1.49
Compute Final Result	0.04
Overall	2.43

images can be found by the following function:

$$L_N = N * 2.43$$

Moreover, using this function we can compute the number of face images that can be recognized in real-time, less than 16ms. After simple calculations, we conclude that 6 images can be recognized in the real-time boundary. This can be also seen in Figure 6.5. Although this is not important for the current system, as the specifications indicate a stream of 60fps, it is helpful in comprehending the high performance of the system, which might be useful in future implementations.

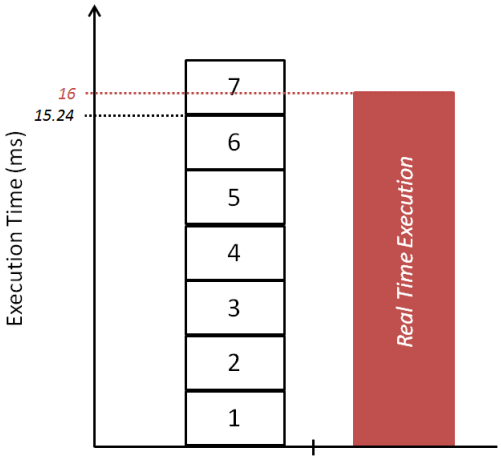

Figure 6.5: *Execution time compared to real-time execution*

In the experiment described in Section 6.1, the process of importing the test images is added to the latency of the system. This latency, after testing, is found to be 14 ms. This latency is added to the execution time of recognition. So every new image is imported and recognized in 16.43 ms. The total execution, for all input images, is seen in Figure 6.6.

To evaluate better the performance of our system, it is compared to similar implementations. Although, such a comparison is not entirely accurate, as there are different parameters utilized for every implementation (feature vector size, image size, etc.), it is adequate to comprehend the level of performance. In Table 6.2, a comparison with implementations that perform face recognition using LBPH, is presented. Those implementations are [8], [10] and the implementation referred as ARM, which is the implementation in the ARM, described in Section 5.1. The proposed implementation performs much better than the other similar ones. The implementation that its performance approaches ours is GPU [8], but we should point out that is applied to smaller images of size $130x150$.

Furthermore, the proposed system is performing better than [9], which utilized LBP for face detection, a similar but simpler task. The latency of that implementation is 16.3 ms, almost 14 ms higher than the proposed implementation. In [9], we find the execution times of two more implementations in CUDA and in Plain C. The execution time for those two is again worse than our system, with 11.4 ms and 125 ms respectively.

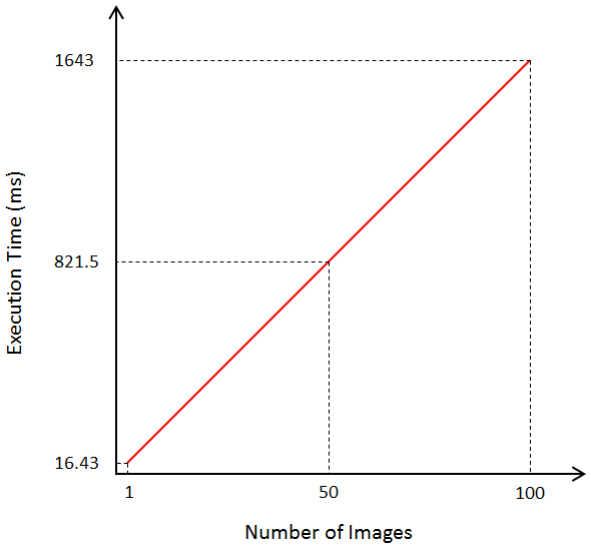

Figure 6.6: *Execution time for 100 images*

Table 6.2: Different implementations comparison

Implemenation	Latency (ms)
OpenCL [10]	25.6
GPU [8]	3.9
GPUX [8]	25.62
ANN [8]	104.69
ARM (5.1)	1414.39
Proposed System	**2.43**

6.3 Accuracy

The accuracy is the next important aspect of our system. In this section, it will be evaluated for the proposed implementation, through a set of experiments described in Section 6.1. As described, 3 sets of experiments will be performed. For each one of them, 5 images of 20 different people will be used to construct our database and 100 test images will be used as an input for recognition. During these experiments, the recognition results are examined and the correct recognitions (CR) are marked. The recognition accuracy for the 3 sets of experiments are presented in Table 6.3. The accuracy for the three different sets are 78%, 81% and 77% respectively, leading to an average accuracy of 78.6% for our system.

Those results regard first rank accuracy. To examine the different ranks' accuracy results, for better overview of our system, a modification on the design in needed. This modification is necessary, as we need, depending on the rank, to observe more distances than the minimum one. This way, we change the Comparison Histogram IP core, to output all the calculated distances. This way we can observe the accuracy for different ranks. This modification on the design was just applied for this experiment and was

Table 6.3: Experiments' Accuracy Results

Set	Faces	CR	Accuracy (%)
1	100	78	78%
2	100	81	81%
3	100	77	77%
Total	300	236	78.6%

discarded after the end of it. The different rank accuracy results are presented in Figure 6.7. We observe that the increase of the rank leads to higher accuracy, with a smaller difference between highest ranks. The current system is using first rank recognition, but once more those results are presented to comprehend the systems efficiency better and point out its possibilities for different implementations.

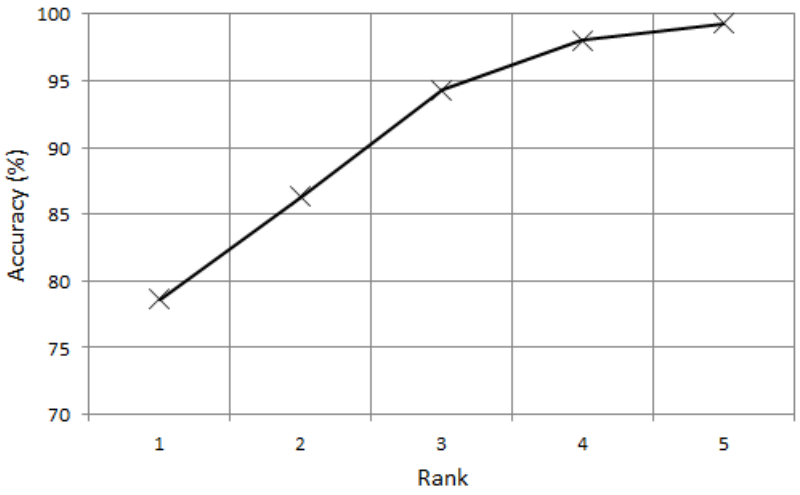

Figure 6.7: *Accuracy for different ranks*

If we refer to [29], which presents a wide range of different accuracy results for face recognition, the high accuracy level of our system can be confirmed. In addition, a comparison with the also high-performance implementation of LBPH in [10], shows that a better accuracy is achieved. The accuracy of 74.75% in [10] is almost 4% worse, which is a notable difference.

6.4 Resource Utilization

The utilization of our system in not a prime concern of our system, as the main goal was to achieve a real-time, high accuracy face recognition system, regardless of the resources utilization. However, the resources utilization is presented to give the reader a complete description of our system.

Regarding the platform [14], one ARM processor is utilized for the system in addition to the FPGA. The memory on the platform is of 512 MB, but the developed system

Table 6.4: FPGA Resources Utilization

Resource	Utilization	Available	Utilization (%)
FF	26834	157200	16.78
LUT	34887	78600	44.39
Memory LUT	228	26600	0.86
BRAM	198.5	265	74.91
BUFG	1	32	3.12

requires less than 1.5 MB for its operation. The SD card reader of the FLORIDA was also utilized by our system, during the testing process. As different implementations, will utilized different input means, though, the SD card may not be in need of the system.

The FPGA resources utilization should be examined further, as more important information can be derived from that. The utilization of the main resources of the FPGA, Flip Flops (FF), Look Up Tables (LUT), Memory Look Up Tables (Memory LUT), Block Memory (BRAM) and Global Buffers (BUFG) are presented in Table 6.4. This utilization is also, better, observed through the graph in Figure 6.8. The utilization can be characterized as low, while many resources are still available. On the other hand, compared to the rest of the resources, the BRAM reports a high utilization of 75%. This consents to the fact that the BRAM utilization was a limitation, we came across, many times during the implementation.

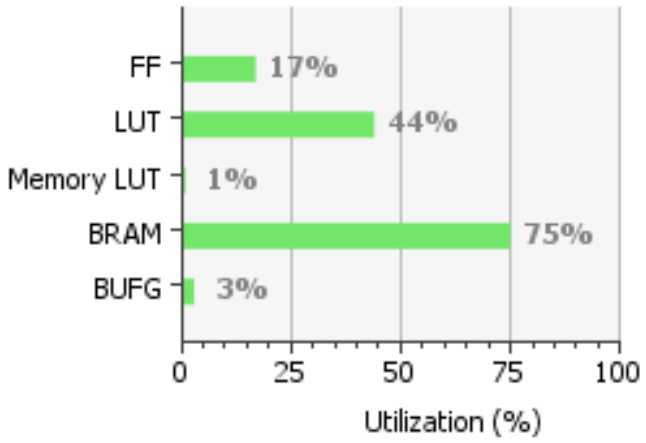

Figure 6.8: *Resource Utilization Percentage*

The utilization of the FPGA resources, can be observed also through the device diagram picture, which is available through Vivado. This overview is presented in Figure 6.9. The programmable logic fabric is delimited by the different color lines and the resources utilized by the light blue-colored areas. We notice that the utilized resources are concentrated in close areas. This is done to minimize the length of the paths that are used.

The resources utilization agrees with the expected estimates presented during the

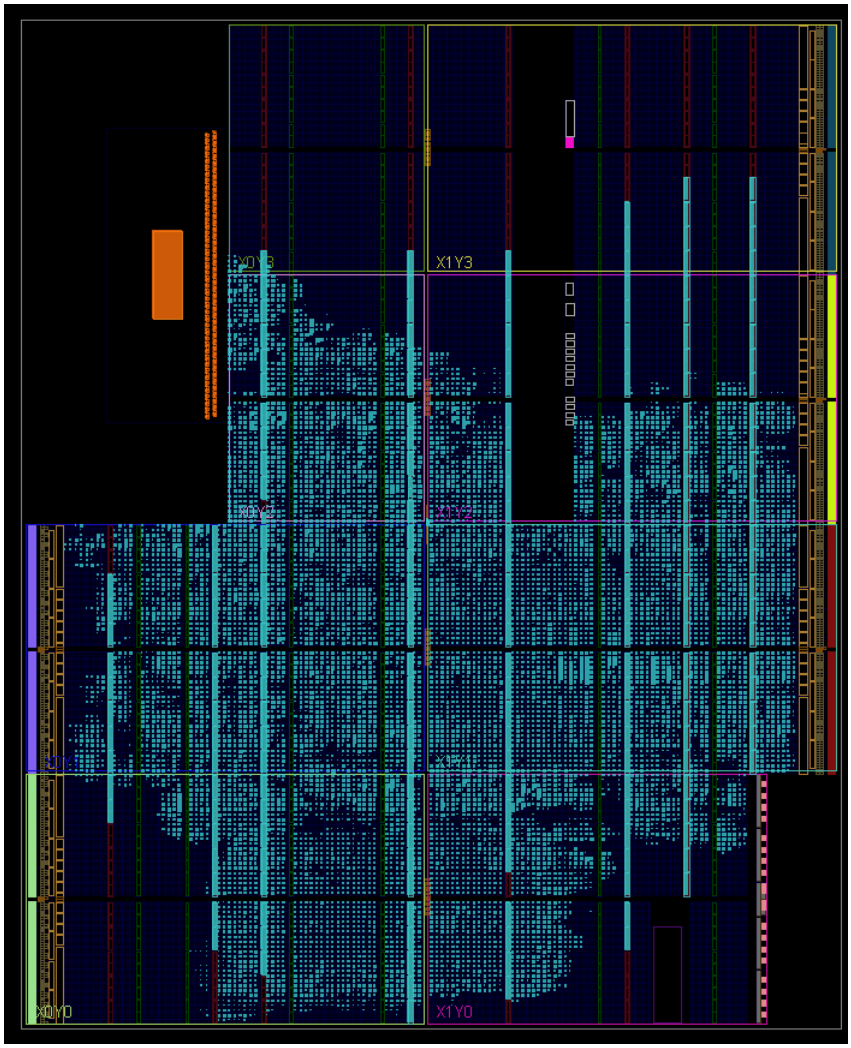

Figure 6.9: *FPGA overview*

implementation. The utilization can be taken in account for future extensions of the current system. An addition of a process in the programmable logic of the system, required by a future implementation, should comply with the available resources and thus should be based on the results presented in this section.

6.5 Power Consumption

The power consumption of our system was not taken into consideration, at any part of the implementation, as there was no limitation on the project specifications. From our point of view, the platform maximum power consumption could be reached, if it was necessary to achieve the necessary performance.

Nevertheless, the power consumption information of a system is very important, especially when we are talking about embedded systems, where power efficiency is some

Table 6.5: Power Consumption Information

Total On-Chip Power (W)	2.536
Dynamic (W)	2.353
Device Static (W)	0.183
Max Ambient (C)	76.6
Junction Temperature (C)	33.4

times mandatory. The proposed implementation, here, may not have been designed with power consumption boundaries, but a future utilization of it in a different, wider, system, which has to comply with power limits, is possible. Hence, we present the basic power consumption information in Table 6.5 and Figure 6.10, acquired by Vivado concerning the FPGA.

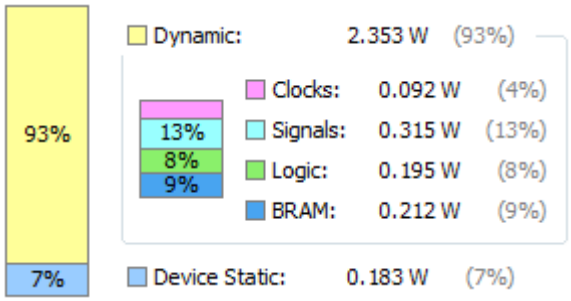

Figure 6.10: *On chip power consumption*

Observing, those results we can notice the low power consumption of the system, mainly when we compare to the other accelerated implementations of LBPH for face recognition [8], [10]. Those implementations utilized GPUs for that purpose, where the power consumption, according to [43], ranges from 50W to 100W. This way, although it was not designed with this aim, in overall, our system falls under the category of a power efficient system.

6.6 Conclusion

The evaluation of our system took place in the current chapter. This evaluation was performed through a set of extensive experiments described in Section 6.1. The results derived from those experiments were examined to verify the efficiency of our system.

The main aim of the implemented system was to achieve real-time, high accuracy face recognition. This goal was achieved successfully with a low execution time and a high accuracy percentage. This can be verified by the results presented in Sections 6.2 and 6.3. Concerning the performance, the system's execution time for recognizing a single face is 2.43 ms. This time is way below the real-time execution of 16ms and also outperforms similar implementations. Regarding the accuracy, a percentage of 78.6% is reported. This is a high accuracy value, especially when we are talking about first rank recognition.

Except from the prime concerns of the system, information regarding other aspects of the system are provided as well. In Sections 6.4 and 6.5 insight on the resources utilization and power consumption, respectively, is provided. The current utilization leaves a notable margin of available resources with only the BRAM resources being almost completely used. In the power consumption part, the approximately 2.5 W required by our system is a low value, which is also lower than similar implementations.

In conclusion, the results presented in this chapter suggest a system that is capable of performing face recognition real-time, with high accuracy and low power consumption. In addition, the remaining available resources allow further extensions and design differentiations for altered system specifications.

7 Conclusion

The main goal of this work, as described in the Introduction, was to implement a system that can perform real-time, high accuracy face recognition. This target was achieved effectively, with the current system meeting those requirements. In addition, this novel system managed to outperform similar implementations and can be characterized, therefore, as a remarkable contribution in the field of facial recognition.

The main conclusions drawn from this work are described in Section 7.1. The main contributions of the current thesis are presented in Section 7.2 and finally, in Section 7.3, recommendations for future research, regarding the current work, are proposed.

7.1 Summary

The thesis is organized in 7 chapters. Excluding the current one and the first chapter which is an introduction to the work carried out, every other chapter is centered on a different aspect of the thesis.

In Chapter 2, background information, essential for comprehending the work carried out, is presented. Basic information on the field of facial recognition is provided. Where is facial recognition used and how a face recognition system works is presented. Furthermore, information concerning reconfigurable computing is given in Chapter 2. As our system is based on reconfigurable computing, basic concepts on the field should be explained. Finally, the platform utilized for our system, Topic Development Kit, is presented and all of its features, that are important to our system, are described.

Chapter 3 consists of a literature review on facial recognition models and similar implementations. Initially, the most popular algorithms that are utilized for face recognition are presented. Those algorithms are: Eigenfaces, Fisherfaces, Elastic Bunch Graph, Local Binary Patterns Histograms and Scale Invariant Feature Transform. The way each algorithm functions and performs recognition, is described. In addition the efficiency and performance of all the algorithms is compared in order to select the most proper one for our system. Through this process, LBPH is chosen for our implementation. Next, also in Chapter 3, implementations that utilize LBPH are presented. Those implementations have a similar aim, as our project, and this is to increase the performance of LBPH utilization. Examining those implementations, we have reported that our system is a novel design, as LBPH was never implemented in the past to perform face recognition by utilizing an FPGA. Also, the other implementations provided a comparison measure, regarding performance, for our implementation.

The complete development of the algorithm has been presented in this Chapter 4. This process was mandatory before proceeding to the system implementation. The proper parameters and models for the system needed to be examined, prior to the implementation, in order to ensure the efficiency of our system. The finalized developed

prototype algorithm set the basis and guidelines for the implementation. The key aspects investigated included parameters of the LBPH algorithm and included models. In Section 4.1, the number of regions of LBPH was decided, assuring high accuracy and relatively small database. Section 4.2 describes the way the proper region weights were chosen to further increase accuracy. Different distance measures, needed for the histogram matching process, were presented in Section 4.3, and the most appropriate one, Manhattan Distance model, was selected. Finally, the utilization of classification and its benefits were described in Section 4.4. To sum up, through a set of experiments, the algorithm for our system was developed following the most appropriate path. This process' goal was to maximize the efficiency of the algorithm, always with respect to the project specifications. This goal is confirmed by the final 78% first rank accuracy achieved and also by ensuring that the model complexity is not burdened.

In Chapter 5, the complete process of implementing our system has been described extensively. In Section 5.1, our design approach was described, including the profiling of the models. After that, the two IP cores needed for our system, Feature Extraction IP and Histograms Comparison IP, were designed and optimized efficiently. The process of doing that for the two cores is desribed in Sections 5.2 and 5.3 respectively. Once the IPs were completely designed, their integration on the PL of the system took place and was described in Section 5.4. Finally, the role of the ARM processor on the operation of the system, although limited, was presented in Section 5.5. The final system designed, through this implementation, met all the requirements that were expected, high accuracy, real-time face recognition.

Finally, the evaluation of our system took place in Chapter 6. This evaluation was performed through a set of extensive experiments described in Section 6.1. The results derived from those experiments were examined to verify the efficiency of our system. The main aim of the implemented system was to achieve real-time, high accuracy face recognition. This goal was achieved successfully with a low execution time and a high accuracy percentage. This can be verified by the results presented in Sections 6.2 and 6.3. Concerning the performance, the system's execution time for recognizing a single face is 2.43 ms. This time is way below the real-time execution of 16 ms and also outperforms similar implementations. Regarding the accuracy, a percentage of 78.6% is reported. This is a high accuracy value, especially when we are talking about first rank recognition. Except from the prime concerns of the system, information regarding other aspects of the system are provided as well. In Sections 6.4 and 6.5 insight on the resources utilization and power consumption, respectively, is provided. The current utilization leaves a notable margin of available resources with only the BRAM resources being almost completely used. In the power consumption part, the approximately 2.5 W required by our system is a low value, which is also lower than similar implementations. To conclude, the results presented in Chapter 6 suggest a system that is capable of performing face recognition real-time, with high accuracy and low power consumption. In addition, the remaining available resources allow further extensions and design differentiations for altered system specifications.

7.2 Main Contributions

The design of this system was carried out with respect to the project specifications, described in Section 1.2. Based on those specifications and through the implementation process an eminent system was developed. The main contributions of this thesis are:

- A system that performs real-time, high accuracy face recognition was implemented. In addition, the performance and accuracy achieved exceed the initial project requirements.
- This thesis presents a novel system for face recognition. It is the first time, as known to the author, that LBPH was implemented on an FPGA to perform face recognition.
- Complete face recognition, with an execution time of 2.43 ms, was achieved. This time meets the requirements, of a real-time execution, which corresponds to less than 16 ms as described in Section 1.3. In addition, this performance outmatches similar implementations on high performance facial recognition.
- The accuracy of the system is higher than 78%. This percentage is congruent with the the first rank accuracy of the most often utilized, high-accuracy, models. Also, the accuracy reported is higher, even up to 4 p.p., than other accelerated face recognition implementations.
- The proposed system requires a power consumption of 2.536 W. Therefore, the proposed system is power efficient, especially when it is compared to face recognition systems, that utilize different means for acceleration, such as GPUs [43].
- The system does not utilize all the available resources of the FPGA platform. 83% of the total FFs, 56% of the total LUTs and 25% of the total BRAM, remain available for utilization. This gives the ability of functional additions in the system, through the utilization of the available resources. Another possibility is to utilize an earlier FPGA model, that can still meet the utilization demands, which would be cost-effective.
- The proposed system is simple in its operation and also allows its utilization for different project specifications easily. As mentioned before, the input and output methods can be simply modified according to the system requirements. Storing the image on the memory of the platform is the only obligation for utilizing the current design, which is easily accessed.
- A product that directly addresses the interests of Topic Embedded Systems' clients was developed. Furthermore, the IP cores designed for the FPGA are now in the possession of the company, for future applications and for further development.

7.3 Future Work

In the current section, proposals for future research are made. Although, this novel proposed system is characterized by its high performance and efficiency, some of its

aspects could be researched further, to improve it more and also to take in account the evolution of technology and FPGAs, in specific. The future work proposals are the following:

- Defined input and output implementations should be investigated. This way, the transfer time of input and output data, could be reduced significantly. For example, a system that utilizes a camera for importing face images (frames), shall now transfer the data to the memory first, while the import of the data directly to the FPGA would be more time-efficient. The Topic Development Kit offers a range of different ports that could be utilized for different input and output methods. Hence, an implementation that can be easily adjusted to utilize them efficiently could be researched in the future.

- The system developed, as described, is running on bare-metal. A future research could focus on booting an operating system on the platform and run the designed application on top of it. The modifications on the implementation should be investigated and the benefits that can come out of it should be highlighted.

- As the technology evolves rapidly, new solutions are available. Concerning the current system, presently manufactured FPGAs offer more resources that can be utilized. A future work could examine the dependency between the performance and the resources available. This way, the current design could be migrated easily and efficiently to brand new hardware.

- Another future research that is interesting, regards the accuracy of our model. Currently, the new recognized images do not interfere with the database. However, the correctly recognized images could be integrated in the database and increase the robustness of recognition. The ratio of accuracy increase to the latency of such a process and in addition how could this be implemented efficiently on the FPGA should be examined.

- Last, modifications on our implementation, focusing on power efficiency, could be researched in the future. The performance of the proposed system is much higher than the real-time goal. Therefore, the available margin in performance degradation, could be exploited to reduce the power consumption. Many systems' main concern is power efficiency and hence, a research on altering the current design towards that goal could be useful.

Bibliography

[1] W. W. Bledsoe, "The model method in facial recognition," *Technical Report PRI 15, Panoramic Research, Inc., Palo Alto, California.*, 1964.

[2] L. D. H. A. J. Goldstein and A. B. Lesk, "Identification of human faces," *Proc. IEEE*, vol. 59, no. 5, pp. 748–760, 1971.

[3] A. P. Matthew Turk, "Eigenfaces for recognition," *Journal of cognitive neuroscience.*, vol. 3, no. 1, pp. 71–86, 1991.

[4] D. J. K. Peter N. Belhumeur, J. P. Hespanha, "Eigenfaces vs. fisherfaces: Recognition using class specific linear projection," *IEEE Transcations on Pattern Analysis and Machine Intelligence*, vol. 19, no. 7, pp. 711–720, 1997.

[5] L. e. a. Wiskott, "Face recognition by elastic bunch graph matching," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, pp. 775–779, 1997.

[6] H. A. Ahonen, T. and M. Pietikinen, "Face description with local binary patterns," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 28, no. 12, pp. 2037–2041, 2006.

[7] D. G. Lowe, "Object recognition from local scale invariant features," *Proc. 7th International Conference on Computer Vision (ICCV'99), Corfu, Greece*, pp. 1150–1157, 1999.

[8] S. C. Tek and M. Gkmen, "Cuda accelerated face recognition using local binary patterns." *Proceedings of Winter Seminar on Computer Graphics*, 2012.

[9] A. H. Roman Jurnek, Pavel Zemck, "Implementing the local binary patterns with simd instructions of cpu," *Proceedings of Winter Seminar on Computer Graphics*, p. 5, 2010.

[10] C. Y. N. Dwith and G. N. Rathna, "Parallel implementation of lbp based face recognition on gpu using opencl," in *2012 13th International Conference on Parallel and Distributed Computing, Applications and Technologies*, Dec 2012, pp. 755–760.

[11] H. S. Compton Katherine, "An introduction to reconfigurable computing," *IEEE Computer*, 2000.

[12] *FPGA Fundementals.* National Instrument, May 2013.

[13] G. Kuzmanov, *Lectures in Reconfigurable Computing Design.* Delft University of Technology, November 2014.

[14] *Topic Development Kit leaflet.* Topic Embedded Systems.

[15] *Zynq-7000 All Programmable SoC Overview (DS190).* Xilinx.

[16] K. Pearson, "On lines and planes of closest fit to systems of points in space," *Philosophical Magazine*, vol. 2, no. 11, pp. 559–572, 1901.

[17] S. Zhang and M. Turk, "Eigenfaces," *Scholarpedia*, vol. 3, no. 9, p. 4244, 2008.

[18] R. A. Fisher, "The use of multiple measures in taxonomic problems," *Annal Eugenics*, vol. 7, pp. 179–188, 1936.

[19] L. Wiskott, R. P. Wrtz, and G. Westphal, "Elastic Bunch Graph Matching," *Scholarpedia*, vol. 9, no. 3, p. 10587, 2014.

[20] L. Wiskott and C. von der Malsburg, "Face recognition by dynamic link matching," *Chapter 11 in Lateral Interactions in the Cortex*, 1996.

[21] P. M. Ojala, T. and D. Harwood, "A comparative study of texture measures with classification based on feature distributions," *Pattern Recognition*, vol. 19, no. 3, pp. 51–59, 1996.

[22] M. Pietikinen, "Local Binary Patterns," *Scholarpedia*, vol. 5, no. 3, p. 9775, 2010.

[23] M. P. T. Ojala and T. Maenpaa, "Multiresolution gray-scale and rotation invariant texture classification with local binary patterns," *Pattern Analysis and Machine Intelligence, IEEE Transactions*, vol. 24, no. 7, pp. 971–987, 2002.

[24] P. O. Due Thanh Nguyen and W. Li, "Human detection with contour-based local motion binary patterns," *Image Processing (ICIP), 2011 18th IEEE International Conference on*, pp. 3609–3612, 2011.

[25] T. Lindeberg, "Scale Invariant Feature Transform," *Scholarpedia*, vol. 7, no. 5, p. 10491, 2012.

[26] L. T., "Scale-space theory: A basic tool for analysing structures at different scales," *Journal of Applied Statistics*, vol. 21, no. 2, pp. 224–270, 1994.

[27] M. Shah, "Computer vision, lecture 5," 2012.

[28] S. S. Raman Kumar, "Performance comparison of eigenfaces vs. fisherfaces for face recognition," *International Journal of Computer and Organization Trends*, vol. 3, no. 8, pp. 314–317, 2013.

[29] "Baseline results on the feret database," 2010. [Online]. Available: http://www.cs.colostate.edu/evalfacerec

[30] S. T. Aruni Singh, Sanjay Kumar Singh, "Comparison of face recognition algorithms on dummy facess," *The International Journal of Multimedia and Its Applications (IJMA)*, vol. 4, no. 4, pp. 121–135, 2012.

[31] N.Lavanyadevi, "Comparison of various algorithms for face matching and retrieval in forensic applications," *International Journal of Engineering Science and Innovative Technology (IJESIT)*, vol. 3, no. 4, pp. 329–335, 2014.

[32] M. G. Tomasz Kryjak, Mateusz Komorkiewicz, "Fpga implementation of real-time head-shoulder detection using local binary patterns, svm and foreground object detection," *Design and Architectures for Signal and Image Processing (DASIP), 2012 Conference on*, pp. 1–8, 2012.

[33] L. Lenc and P. Král, "Unconstrained Facial Images: Database for face recognition under real-world conditions," in *14th Mexican International Conference on Artificial Intelligence (MICAI 2015)*. Cuernavaca, Mexico: Springer, 25-31 October 2015 2015.

[34] C. H. Chan, "Multi-scale local binary pattern histogram for face recognition," *University of Surrey*, 2008.

[35] P. K., "On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling," *Phil. Mag*, vol. 5, no. 50, pp. 157–175, 1900.

[36] E. F. Krause, *Taxicab Geometry: An Adventure in Non-Euclidean Geometry*, 1986.

[37] T. W. M. S. A. Md. Abdur Rahim, Md. Najmul Hossain, "Face recognition using local binary patterns (lbp)," *Global Journal of Computer Science and Technology Graphics and Vision*, vol. 13, no. 4, 2013.

[38] D. v. d. H. Nikolaos Stekas, "Face recognition using local binary patterns histograms (lbph) on an fpga-based system on chip (soc)," *30th IEEE IPDPS, Reconfigurable Architectures Workshop*, May 2016.

[39] *AXI Reference Guide (UG761)*. Xilinx, 2011.

[40] *AXI DMA v7.1 LogiCORE IP Product Guide (PG021)*. Xilinx, 2015.

[41] *Block Memory Generator v8.2 LogiCORE IP Product Guide (PG058)*. Xilinx, 2015.

[42] *AXI Interconnect v2.1 LogiCORE IP Product Guide (PG059)*. Xilinx, 2015.

[43] S. Collange, D. Defour, and A. Tisserand, "Power consumption of gpus from a software perspective," in *Proceedings of the 9th International Conference on Computational Science: Part I*, 2009, pp. 914–923.